The seqinr Package

October 27, 2008

Version 2.0-0

Date 2008-10-26

Title Biological Sequences Retrieval and Analysis

Author Delphine Charif and Jean R. Lobry and Anamaria Necsulea and Leonor Palmeira and Simon Penel

Maintainer Simon Penel <penel@biomserv.univ-lyon1.fr>

Depends R (>= 2.5.0)

Suggests ade4, segmented

Description Exploratory data analysis and data visualization for biological sequence (DNA and protein) data. Include also utilities for sequence data management under the ACNUC system.

License GPL (>= 2)

URL http://seqinr.r-forge.r-project.org/, Mailing list: http://r-forge.r-project.org/mail/?group_id=220

ZipData no

R topics documented:

`AAstat` *To Get Some Protein Statistics*

Description

Returns simple protein sequence information including the number of residues, the percentage physico-chemical classes and the theoretical isoelectric point.

Usage

```
AAstat(seq, plot = TRUE)
```

Arguments

`seq`	a protein sequence as a vector of upper-case chars
`plot`	if `TRUE`, plots the presence of residues splited by physico-chemical classes along the sequence.

Value

A list with the three following components:

`Compo`	A factor giving the amino acid counts.
`Prop`	A list giving the percentage of each physico-chemical classes (Tiny, Small, Aliphatic, Aromatic, Non-polar, Polar, Charged, Positive, Negative).
`Pi`	The theoretical isoelectric point

Author(s)

D. Charif, J.R. Lobry

References

```
citation("seqinr")
```

See Also

`computePI`, `SEQINR.UTIL`, `SeqFastaAA`

Examples

```
seqAA <- read.fasta(file = system.file("sequences/seqAA.fasta", package = "seqinr"), seqty
AAstat(seqAA[[1]])
```

AnoukResult — *Expected numeric results for Ka and Ks computation*

Description

This data set is what should be obtained when runing `kaks()` on the test file Anouk.fasta in the sequences directory of the seqinR package.

Usage

```
data(AnoukResult)
```

Format

A list with 4 components of class dist.

ka Ka

ks Ks

vka variance for Ka

vks variance for Ks

Details

See the example in `kaks`.

Source

The fasta test file was provided by Anamaria Necsulea.

References

```
citation("seqinr")
```

EXP — *Vectors of coefficients to compute linear forms.*

Description

This dataset is used to compute linear forms on codon frequencies: if `codfreq` is a vector of codon frequencies then `drop(freq %*% EXP$CG3)` will return for instance the G+C content in third codon positions. Base order is the lexical order: a, c, g, t (or u).

Usage

```
data(EXP)
```

Format

List of 24 vectors of coefficients

A num [1:4] 1 0 0 0

A3 num [1:64] 1 0 0 0 1 0 0 0 1 0 ...

AGZ num [1:64] 0 0 0 0 0 0 0 0 1 0 ...

ARG num [1:64] 0 0 0 0 0 0 0 0 1 0 ...

AU3 num [1:64] 1 0 0 1 1 0 0 1 1 0 ...

BC num [1:64] 0 1 0 0 0 0 0 0 0 0 ...

C num [1:4] 0 1 0 0

C3 num [1:64] 0 1 0 0 0 1 0 0 0 1 ...

CAI num [1:64] 0.00 0.00 -1.37 -2.98 -2.58 ...

CG num [1:4] 0 1 1 0

CG1 num [1:64] 0 0 0 0 0 0 0 0 0 0 ...

CG12 num [1:64] 0 0 0 0 0.5 0.5 0.5 0.5 0.5 0.5 ...

CG2 num [1:64] 0 0 0 0 1 1 1 1 1 1 ...

CG3 num [1:64] 0 1 1 0 0 1 1 0 0 1 ...

CGN num [1:64] 0 0 0 0 0 0 0 0 0 0 ...

F1 num [1:64] 1.026 0.239 1.026 0.239 -0.097 ...

G num [1:4] 0 0 1 0

G3 num [1:64] 0 0 1 0 0 0 1 0 0 0 ...

KD num [1:64] -3.9 -3.5 -3.9 -3.5 -0.7 -0.7 -0.7 -0.7 -4.5 -0.8 ...

Q num [1:64] 0 0 0 0 1 1 1 1 0 0 ...

QA3 num [1:64] 0 0 0 0 1 0 0 0 0 0 ...

QC3 num [1:64] 0 0 0 0 0 1 0 0 0 0 ...

U num [1:4] 0 0 0 1

U3 num [1:64] 0 0 0 1 0 0 0 1 0 0 ...

Details

It's better to work directly at the amino-acid level when computing linear forms on amino-acid frequencies so as to have a single coefficient vector. For instance `EXP$KD` to compute the Kyte and Doolittle hydrophaty index from codon frequencies is valid only for the standard genetic code.

An alternative for `drop(freq %*% EXP$CG3)` is `sum( freq * EXP$CG3 )`, but this is less efficient in terms of CPU time. The advantage of the latter, however, is that thanks to recycling rules you can use either `sum( freq * EXP$A )` or `sum( freq * EXP$A3 )`. To do the same with the %*% operator you have to explicit the recycling rule as in `drop( freq %*% rep(EXP$A, 16))`.

Source

ANALSEQ EXPFILEs for command EXP.
http://biomserv.univ-lyon1.fr/doclogi/docanals/manuel.html

References

```
citation("seqinr")
```

A content in A nucleotide

A3 content in A nucleotide in third position of codon

AGZ Arg content (aga and agg codons)

ARG Arg content

AU3 content in A and U nucleotides in third position of codon

BC Good choice (Bon choix). Gouy M., Gautier C. (1982) codon usage in bacteria : Correlation with gene expressivity. *Nucleic Acids Research*,**10(22)**:7055-7074.

C content in C nucleotides

C3 content in A nucleotides in third position of codon

CAI Codon adaptation index for E. coli. Sharp, P.M., Li, W.-H. (1987) The codon adaptation index - a measure of directionam synonymous codon usage bias, and its potential applications. *Nucleic Acids Research*,**15**:1281-1295.

CG content in G + C nucleotides

CG1 content in G + C nucleotides in first position of codon

CG12 content in G + C nucleotides in first and second position of codon

CG2 content in G + C nucleotides in second position of codon

CG3 content in G + C nucleotides in third position of codon

CGN content in CGA + CGU + CGA + CGG

F1 From Table 2 in Lobry, J.R., Gautier, C. (1994) Hydrophobicity, expressivity and aromaticity are the major trends of amino-acid usage in 999 *Escherichia coli* chromosome-encode genes. *Nucleic Acids Research*,**22**:3174-3180.

G3 content in G nucleotides in third position of codon

KD Kyte, J., Doolittle, R.F. (1982) A simple method for displaying the hydropathic character of a protein. *J. Mol. Biol.*,**157** :105-132.

Q content in quartet

QA3 content in quartet with the A nucleotide in third position

QC3 content in quartet with the A nucleotide in third position

U content in U nucleotide

U3 content in U nucleotides in third position of codon

Examples

```
data(EXP)
```

G+C Content *Calculates the fractional G+C content of nucleic acid sequences.*

Description

Calculates the fraction of G+C bases of the input nucleic acid sequence(s). It reads in nucleic acid sequences, sums the number of 'g' and 'c' bases and writes out the result as the fraction (in the interval 0.0 to 1.0) to the total number of 'a', 'c', 'g' and 't' bases. Global G+C content `GC`, G+C in the first position of the codon bases `GC1`, G+C in the second position of the codon bases `GC2`, and G+C in the third position of the codon bases `GC3` can be computed. All functions can take ambiguous bases into account when requested.

Usage

```
GC(seq, forceToLower = TRUE, exact = FALSE, NA.GC = NA, oldGC = FALSE)
GC1(seq, frame = 0, ...)
GC2(seq, frame = 0, ...)
GC3(seq, frame = 0, ...)
GCpos(seq, pos, frame = 0, ...)
```

Arguments

`seq`	a nucleic acid sequence as a vector of single characters
`frame`	for coding sequences, an integer (0, 1, 2) giving the frame
`forceToLower`	logical. if `TRUE` force sequence characters in lower-case. Turn this to `FALSE` to save time if your sequence is already in lower-case (cpu time is approximately divided by 3 when turned off)
`exact`	logical: if `TRUE` ambiguous bases are taken into account when computing the G+C content (see details). Turn this to `FALSE` to save time if your you can neglect ambiguous bases in your sequence (cpu time is approximately divided by 3 when turned off)
`NA.GC`	what should be returned when the GC is impossible to compute from data, for instance with NNNNNNN. This behaviour could be different when argument `exact` is `TRUE`, for instance the G+C content of WWSS is `NA` by default, but is 0.5 when `exact` is set to `TRUE`
`...`	arguments passed to the function `GC`
`pos`	for coding sequences, the codon position (1, 2, 3) that should be taken into account to compute the G+C content
`oldGC`	logical defaulting to `FALSE`: should the GC content computed as in seqinR <= 1.0-6, that is as the sum of 'g' and 'c' bases divided by the length of the sequence. As from seqinR >= 1.1-3, this argument is deprecated and a warning is issued.

Details

When `exact` is set to `TRUE` the G+C content is estimated with ambiguous bases taken into account. Note that this is time expensive. A first pass is made on non-ambiguous bases to estimate the probabilities of the four bases in the sequence. They are then used to weight the contributions of ambiguous bases to the G+C content. Let note nx the total number of base 'x' in the sequence. For instance suppose that there are nb bases 'b'. 'b' stands for "not a", that is for 'c', 'g' or 't'. The contribution of 'b' bases to the GC base count will be:

nb*(nc + ng)/(nc + ng + nt)

The contribution of 'b' bases to the AT base count will be:

nb*nt/(nc + ng + nt)

All ambiguous bases contributions to the AT and GC counts are weighted is similar way and then the G+C content is computed as ngc/(nat + ngc).

Value

`GC` returns the fraction of G+C (in [0,1]) as a numeric vector of length one. `GCpos` returns GC at position `pos`. `GC1`, `GC2`, `GC3` are wrappers for `GCpos` with the argument `pos` set to 1, 2, and 3, respectively. `NA` is returned when `seq` is `NA`. `NA.GC` defaulting to `NA` is returned when the G+C content can not be computed from data.

Author(s)

D. Charif and L. Palmeira and J.R. Lobry

References

`citation("seqinr")`.

The program codonW used here for comparison is available at `http://codonw.sourceforge.net/`.

See Also

You can use `s2c` to convert a string into a vetor of single character and `tolower` to convert upper-case characters into lower-case characters. Do not confuse with `gc` for garbage collection.

Examples

```
   mysequence <- s2c("agtctgggggcccctttaagtagatagatagctagtcgta")
   GC(mysequence)  # 0.4761905
   GC1(mysequence) # 0.6428571
   GC2(mysequence) # 0.3571429
   GC3(mysequence) # 0.4285714
#
# With upper-case characters:
#
  myUCsequence <- s2c("GGGGGGGGGA")
  GC(myUCsequence) # 0.9
#
```

```
# With ambiguous bases:
#
  GC(s2c("acgt")) # 0.5
  GC(s2c("acgtssss")) # 0.5
  GC(s2c("acgtssss"), exact = TRUE) # 0.75
#
# Missing data:
#
  stopifnot(is.na(GC(s2c("NNNN"))))
  stopifnot(is.na(GC(s2c("NNNN"), exact = TRUE)))
  stopifnot(is.na(GC(s2c("WWSS"))))
  stopifnot(GC(s2c("WWSS"), exact = TRUE) == 0.5)
#
# Coding sequences tests:
#
  cdstest <- s2c("ATGATG")
  stopifnot(GC3(cdstest) == 1)
  stopifnot(GC2(cdstest) == 0)
  stopifnot(GC1(cdstest) == 0)
#
# How to reproduce the results obtained with the C program codonW
# version 1.4.4 writen by John Peden. We use here the "input.dat"
# test file from codonW (there are no ambiguous base in these
# sequences).
#
  inputdatfile <- system.file("sequences/input.dat", package = "seqinr")
  input <- read.fasta(file = inputdatfile) # read the FASTA file
  inputoutfile <- system.file("sequences/input.out", package = "seqinr")
  input.res <- read.table(inputoutfile, header = TRUE) # read codonW result file
#
# remove stop codon before computing G+C content (as in codonW)
#
  GC.codonW <- function(dnaseq, ...){
         GC(dnaseq[seq_len(length(dnaseq) - 3)], ...)
  }
  input.gc <- sapply(input, GC.codonW, forceToLower = FALSE)
  max(abs(input.gc - input.res$GC)) # 0.0004946237

  plot(x = input.gc, y = input.res$GC, las = 1,
  xlab = "Results with GC()", ylab = "Results from codonW",
  main = "Comparison of G+C content results")
  abline(c(0, 1), col = "red")
  legend("topleft", inset = 0.01, legend = "y = x", lty = 1, col = "red")
## Not run:
# Too long for routine check
# This is a benchmark to compare the effect of various parameter
# setting on computation time
n <- 10
from <-10^4
to <- 10^5
size <- seq(from = from, to = to, length = n)
res <- data.frame(matrix(NA, nrow = n, ncol = 5))
colnames(res) <- c("size", "FF", "FT", "TF", "TT")
```

```
res[, "size"] <- size

for(i in seq_len(n)){
  myseq <- sample(x = s2c("acgtws"), size = size[i], replace = TRUE)
  res[i, "FF"] <- system.time(GC(myseq, forceToLower = FALSE, exact = FALSE))[3]
  res[i, "FT"] <- system.time(GC(myseq, forceToLower = FALSE, exact = TRUE))[3]
        res[i, "TF"] <- system.time(GC(myseq, forceToLower = TRUE, exact = FALSE))[3]
        res[i, "TT"] <- system.time(GC(myseq, forceToLower = TRUE, exact = TRUE))[3]
}

par(oma = c(0,0,2.5,0), mar = c(4,5,0,2) + 0.1, mfrow = c(2, 1))
plot(res$size, res$TT, las = 1,
xlab = "Sequence size [bp]",
ylim = c(0, max(res$TT)), xlim = c(0, max(res$size)), ylab = "")
title(ylab = "Observed time [s]", line = 4)
abline(lm(res$TT~res$size))
points(res$size, res$FT, col = "red")
abline(lm(res$FT~res$size), col = "red", lty = 3)
points(res$size, res$TF, pch = 2)
abline(lm(res$TF~res$size))
points(res$size, res$FF, pch = 2, col = "red")
abline(lm(res$FF~res$size), lty = 3, col = "red")

legend("topleft", inset = 0.01, legend = c("forceToLower = TRUE", "forceToLower = FALSE"), c
legend("bottomright", inset = 0.01, legend = c("exact = TRUE", "exact = FALSE"),
pch = c(1,2))

mincpu <- lm(res$FF~res$size)$coef[2]

barplot(
c(lm(res$FF~res$size)$coef[2]/mincpu,
  lm(res$TF~res$size)$coef[2]/mincpu,
  lm(res$FT~res$size)$coef[2]/mincpu,
  lm(res$TT~res$size)$coef[2]/mincpu),
horiz = TRUE, xlab = "Increase of CPU time",
col = c("red", "black", "red", "black"),
names.arg = c("(F,F)", "(T,F)", "(F,T)", "(T,T)"), las = 1)
title(ylab = "forceToLower,exact", line = 4)

mtext("CPU time as function of options", outer = TRUE, line = 1, cex = 1.5)
## End(Not run)
```

SEQINR.UTIL *utility data for seqinr*

Description

This data set gives the genetics code, the name of each codon, the IUPAC one-letter code for aminoacids and the physico-chemical class of amino acid and the pK values of amino acids described in Bjellqvist *et al.* (1993).

Usage

```
data(SEQINR.UTIL)
```

Format

`SEQINR.UTIL` is a list containing the 4 following objects:

CODES.NCBI is a data frame containing the genetics code : The standard ('Universal') genetic code with a selection of non-standard codes.

CODON.AA is a three columns data frame. The first column is a factor containing the codon. The second column is a factor giving the aminoacids names for each codon. The last column is a factor giving the IUPAC one-letter code for aminoacids

AA.PROPERTY is a list giving the physico-chemical class of amino acid. The differents classes are the following one : Tiny, Small, Aliphatic, Aromatic, Non.polar, Polar, Charged, Basic, Acidic

pK is a data frame. It gives the pK values of amino acids described in Bjellqvist *et al.* (1993) , which were defined by examining polypeptide migration between pH 4.5 to 7.3 in an immobilised pH gradient gel environment with 9.2M and 9.8M urea at 15 degree or 25 degree

Source

Data prepared by D.Charif ⟨charif@biomserv.univ-lyon1.fr⟩.
The genetic codes have been taken from the ncbi taxonomy database: http://www.ncbi.nlm.nih.gov/Taxonomy/Utils/wprintgc.cgi?mode=c. Last update October 05, 2000.
The IUPAC one-letter code for aminoacids is descibed at: http://www.chem.qmul.ac.uk/iupac/AminoAcid/. pK values of amino acids were taken from Bjellqvist et al.
Bjellqvist, B.,Hughes, G.J., Pasquali, Ch., Paquet, N., Ravier, F., Sanchez, J.-Ch., Frutiger, S. & Hochstrasser, D.F.(1993) The focusing positions of polypeptides in immobilized pH gradients can be predicted from their amino acid sequences. *Electrophoresis*, **14**, 1023-1031.

References

```
citation("seqinr")
```

Examples

```
data(SEQINR.UTIL)
```

SeqAcnucWeb *Sequence coming from a remote ACNUC data base*

Description

`as.SeqAcnucWeb` is called by many functions, for instance by `query`, and should not be directly called by the user. It creates an object of class `SeqAcnucWeb`. `is.SeqAcnucWeb` returns TRUE if the object is of class `SeqAcnucWeb`.

Usage

```
as.SeqAcnucWeb(object, length, frame, ncbigc)
is.SeqAcnucWeb(object)
```

Arguments

object	a string giving the name of a sequence present in the data base
length	a string giving the length of the sequence present in the data base
frame	a string giving the frame of the sequence present in the data base
ncbigc	a string giving the ncbi genetic code of the sequence present in the data base

Value

as.SeqAcnucWeb returns an object sequence of class SeqAcnucWeb. Note that as from seqinR 1.1-3 the slot socket has been deleted to save space for long lists.

Author(s)

D. Charif, J.R. Lobry

References

```
citation("seqinr")
```

Examples

```
  ## Not run:
# Need internet connection
  choosebank("emblTP")
  query("mylist", "sp=felis catus et t=cds et o=mitochondrion")
  stopifnot(is.SeqAcnucWeb(mylist$req[[1]]))
  closebank()
## End(Not run)
```

SeqFastaAA *AA sequence in Fasta Format*

Description

as.SeqFastaAA is called by the function as read.fasta. It creates an object of class SeqFastaAA. is.SeqFastaAA returns TRUE if the object is of class SeqFastaAA. summary.SeqFastaAA gives the AA composition of an object of class SeqFastaAA.

Usage

```
as.SeqFastaAA(object, name = NULL, Annot = NULL)
is.SeqFastaAA(object)
## S3 method for class 'SeqFastaAA':
summary(object,...)
```

Arguments

object	a vector of chars representing a biological sequence
name	NULL a character string specifying a name for the sequence
Annot	NULL a character string specifying some annotations for the sequence
...	additional arguments affecting the summary produced

Value

as.SeqFastaAA returns an object sequence of class SeqFastaAA. summary.SeqFastaAA returns a list which the following components:

composition	the AA counting of the sequence
AA.Property	the percentage of each group of amino acid in the sequence. By example, the groups are small, tiny, aliphatic, aromatic ...

Author(s)

D. Charif

References

```
citation("seqinr")
```

Examples

```
s <- read.fasta(file = system.file("sequences/seqAA.fasta", package = "seqinr"), seqtype="
is.SeqFastaAA(s[[1]])
summary(s[[1]])
myseq <- s2c("MSPTAYRRGSPAFLV*")
as.SeqFastaAA(myseq, name = "myseq", Annot = "blablabla")
myseq
```

SeqFastadna *Class for DNA sequence in Fasta Format*

Description

as.SeqFastadna is called by many functions as read.fasta. It creates an object of class SeqFastadna. is.SeqFastadna returns TRUE if the object is of class SeqFastadna. summary.SeqFastadna gives the base composition of an object of class SeqFastadna.

Usage

```
as.SeqFastadna(object, name = NULL, Annot = NULL)
is.SeqFastadna(object)
## S3 method for class 'SeqFastadna':
summary(object, alphabet = s2c("acgt"), ...)
```

Arguments

object	a vector of chars representing a biological sequence
name	NULL a character string specifying a name for the sequence
Annot	NULL a character string specifying some annotations for the sequence
...	additional arguments affecting the summary produced
alphabet	a vector of single characters

Value

as.SeqFastadna returns an object sequence of class SeqFastadna. summary.SeqFastadna returns a list which the following components:

length	the legth of the sequence
compo	the base counting of the sequence
GC	the percentage of G+C in the sequence

Author(s)

D. Charif

References

```
citation("seqinr")
```

Examples

```
s <- read.fasta(system.file("sequences/malM.fasta",package="seqinr"))
is.SeqFastadna(s[[1]])
summary(s[[1]])
myseq <- s2c("acgttgatgctagctagcatcgat")
as.SeqFastadna(myseq, name = "myseq", Annot = "blablabla")
myseq
```

SeqFrag *Class for sub-sequences*

Description

as.SeqFrag is called by all methods of getFrag, but not directly by the users. It creates an object sequence of class SeqFrag.

Usage

```
as.SeqFrag(object, begin, end, name)
is.SeqFrag(object)
```

Arguments

object	an object sequence of class seqFastadna, seqFastaAA, seqAcnucWeb or seqFrag
begin	the first base of the fragment to get
end	the last base of the fragment to get
name	the name of the sequence

Value

as.SeqFrag returns a biological sequence with the following attributes:

seqMother	the name of the sequence from which the sequence comes
begin	the position of the first base of the fragment on the mother sequence
end	the position of the last base of the fragment on the mother sequence
class	SeqFrag which is the classfor sub-sequence

is.SeqFrag returns TRUE if the object is of class Seqfrag.

Author(s)

D. Charif, J.R. Lobry

References

```
citation("seqinr")
```

See Also

getFrag, getLength, getName, getSequence, getTrans

Examples

```
s <- read.fasta(file = system.file("sequences/malM.fasta", package = "seqinr"))
getFrag(s[[1]], 1, 10)
```

---

a *Converts amino-acid three-letter code into the one-letter one*

Description

This is a vectorized function to convert three-letters amino-acid code into the one-letter one, for instance "Ala" into "A".

Usage

```
a(aa)
```

Arguments

aa	A vector of string. All strings are 3 chars long.

Details

Allowed character values for aa are given by aaa(). All other values will generate a warning and return NA. Called without arguments, a() returns the list of all possible output values.

Value

A vector of single characters.

Author(s)

D. Charif, J.R. Lobry

References

The IUPAC one-letter code for aminoacids is described at: http://www.chem.qmul.ac.uk/iupac/AminoAcid/
citation("seqinr")

See Also

aaa, translate

Examples

```
#
# Show all possible input values:
#

aaa()

#
# Convert them in one letter-code:
#

a(aaa())

#
# Check consistency of results:
#

stopifnot( aaa(a(aaa())) == aaa())

#
# Show what happens with non-allowed values:
#

a("SOS") # should be NA and a warning is generated
```

aaa *Converts amino-acid one-letter code into the three-letter one*

Description

This is a vectorized function to convert one-letter amino-acid code into the three-letter one, for instance "A" into "Ala".

Usage

```
aaa(aa)
```

Arguments

aa	A vector of single characters.

Details

Allowed character values for aa are given by a(). All other values will generate a warning and return NA. Called without arguments, aaa() returns the list of all possible output values.

Value

A vector of char string. All strings are 3 chars long.

Author(s)

J.R. Lobry

References

The IUPAC one-letter code for aminoacids is described at: http://www.chem.qmul.ac.uk/iupac/AminoAcid/ citation("seqinr")

See Also

a, translate

Examples

```
#
# Show all possible input values:
#

a()

#
# Convert them in one letter-code:
#
```

```
aaa(a())

#
# Check consistency of results:
#

stopifnot(a(aaa(a())) == a())

#
# Show what happens with non-allowed values:
#

aaa("Z") # should be NA and a warning is generated
```

`aacost` *Aerobic cost of amino-acids in Escherichia coli and G+C classes*

Description

The metabolic cost of amino-acid biosynthesis in *E. coli* under aerobic conditions from table 1 in Akashi and Gojobori (2002). The G+C classes are from Lobry (1997).

Usage

```
data(aacost)
```

Format

A data frame with 20 rows for the amino-acids and the following 7 columns:

aaa amino-acid (three-letters code).

a amino-acid (one-letter code).

prec precursor metabolites (see details).

p number of high-energy phosphate bonds contained in ATP and GTP molecules.

h number of available hydrogen atoms carried in NADH, NADPH, and FADH2 molcules.

tot total metabolic cost assuming 2 high-energy phosphate bonds per hydrogen atom.

gc an ordered factor (l<m<h) for the G+C class of the amino-acid (see details)

Details

Precursor metabolites are: penP, ribose 5-phosphate; PRPP, 5-phosphoribosyl pyrophosphate; eryP, erythrose 4-phosphate; 3pg, 3-phosphoglycerate; pep, phospho*enol*pyruvate; pyr, pyruvate; acCoA, acetyl-CoA; akg, alpha-ketoglutarate; oaa, oxaloacetate. Negative signs on precursor metabolites indicate chemicals *gained* through biosynthetic pathways. Costs of precursors reflect averages for growth on glucose, acetate, and malate (see Table 6 in the supporting information from Akashi and

Gojobori 2002).

The levels l<m<h for the `gc` ordered factor stand for Low G+C, Middle G+C, High G+C amino-acid, respectively. The frequencies of Low G+C amino-acids monotonously decrease with G+C content. The frequencies of High G+C amino- acids monotonously increase with G+C content. The frequencies of Middle G+C amino-acids first increase and then decrease with G+C content. These G+C classes are from Lobry (1997).

`example(aacost)` reproduces figure 2 from Lobry (2004).

Source

Akashi, H, Gojobori, T. (2002) Metabolic efficiency and amino acid composition in the proteomes of *Escherichia coli* and *Bacillus subtilis*. *Proceedings of the National Academy of Sciences of the United States of America*, **99**:3695-3700.

Lobry, J.R. (1997) Influence of genomic G+C content on average amino-acid composition of proteins from 59 bacterial species. *Gene*, **205**:309-316.

Lobry, J.R. (2004) Life history traits and genome structure: aerobiosis and G+C content in bacteria. *Lecture Notes in Computer Sciences*, **3039**:679-686.

References

```
citation("seqinr")
```

Examples

```
data(aacost)
levels(aacost$gc) <- c("low G+C", "mid G+C", "high G+C")
stripchart(aacost$tot~aacost$gc, pch = 19, ylim = c(0.5,3.5),
     xlim = c(0, max(aacost$tot)),
     xlab = "Metabolic cost (high-energy phosphate bonds equivalent)",
     main = "Metabolic cost of the 20 amino-acids\nas function of their G+C class" )
boxplot(aacost$tot~aacost$gc, horizontal = TRUE, add = TRUE)
```

---

aaindex — *List of 544 physicochemical and biological properties for the 20 amino-acids*

Description

Data were imported from release 9.1 (AUG 2006) of the aaindex1 database. See the reference section to cite this database in a publication.

Usage

```
data(aaindex)
```

Format

A named list with 544 elements having each the following components:

H String: Accession number in the aaindex database.

D String: Data description.

R String: LITDB entry number.

A String: Author(s).

T String: Title of the article.

J String: Journal reference and comments.

C String: Accession numbers of similar entries with the correlation coefficients of 0.8 (-0.8) or more (less). Notice: The correlation coefficient is calculated with zeros filled for missing values.

I Numeric named vector: amino acid index data.

Details

A short description of each entry is available under the D component:

alpha-CH chemical shifts (Andersen et al., 1992)
Hydrophobicity index (Argos et al., 1982)
Signal sequence helical potential (Argos et al., 1982)
Membrane-buried preference parameters (Argos et al., 1982)
Conformational parameter of inner helix (Beghin-Dirkx, 1975)
Conformational parameter of beta-structure (Beghin-Dirkx, 1975)
Conformational parameter of beta-turn (Beghin-Dirkx, 1975)
Average flexibility indices (Bhaskaran-Ponnuswamy, 1988)
Residue volume (Bigelow, 1967)
Information value for accessibility; average fraction 35 Information value for accessibility; average fraction 23 Retention coefficient in TFA (Browne et al., 1982)
Retention coefficient in HFBA (Browne et al., 1982)
Transfer free energy to surface (Bull-Breese, 1974)
Apparent partial specific volume (Bull-Breese, 1974)
alpha-NH chemical shifts (Bundi-Wuthrich, 1979)
alpha-CH chemical shifts (Bundi-Wuthrich, 1979)
Spin-spin coupling constants 3JHalpha-NH (Bundi-Wuthrich, 1979)
Normalized frequency of alpha-helix (Burgess et al., 1974)
Normalized frequency of extended structure (Burgess et al., 1974)
Steric parameter (Charton, 1981)
Polarizability parameter (Charton-Charton, 1982)
Free energy of solution in water, kcal/mole (Charton-Charton, 1982)
The Chou-Fasman parameter of the coil conformation (Charton-Charton, 1983)
A parameter defined from the residuals obtained from the best correlation of the Chou-Fasman parameter of beta-sheet (Charton-Charton, 1983)
The number of atoms in the side chain labelled 1+1 (Charton-Charton, 1983)
The number of atoms in the side chain labelled 2+1 (Charton-Charton, 1983)
The number of atoms in the side chain labelled 3+1 (Charton-Charton, 1983)

The number of bonds in the longest chain (Charton-Charton, 1983)
A parameter of charge transfer capability (Charton-Charton, 1983)
A parameter of charge transfer donor capability (Charton-Charton, 1983)
Average volume of buried residue (Chothia, 1975)
Residue accessible surface area in tripeptide (Chothia, 1976)
Residue accessible surface area in folded protein (Chothia, 1976)
Proportion of residues 95 Proportion of residues 100 Normalized frequency of beta-turn (Chou-Fasman, 1978a)
Normalized frequency of alpha-helix (Chou-Fasman, 1978b)
Normalized frequency of beta-sheet (Chou-Fasman, 1978b)
Normalized frequency of beta-turn (Chou-Fasman, 1978b)
Normalized frequency of N-terminal helix (Chou-Fasman, 1978b)
Normalized frequency of C-terminal helix (Chou-Fasman, 1978b)
Normalized frequency of N-terminal non helical region (Chou-Fasman, 1978b)
Normalized frequency of C-terminal non helical region (Chou-Fasman, 1978b)
Normalized frequency of N-terminal beta-sheet (Chou-Fasman, 1978b)
Normalized frequency of C-terminal beta-sheet (Chou-Fasman, 1978b)
Normalized frequency of N-terminal non beta region (Chou-Fasman, 1978b)
Normalized frequency of C-terminal non beta region (Chou-Fasman, 1978b)
Frequency of the 1st residue in turn (Chou-Fasman, 1978b)
Frequency of the 2nd residue in turn (Chou-Fasman, 1978b)
Frequency of the 3rd residue in turn (Chou-Fasman, 1978b)
Frequency of the 4th residue in turn (Chou-Fasman, 1978b)
Normalized frequency of the 2nd and 3rd residues in turn (Chou-Fasman, 1978b)
Normalized hydrophobicity scales for alpha-proteins (Cid et al., 1992)
Normalized hydrophobicity scales for beta-proteins (Cid et al., 1992)
Normalized hydrophobicity scales for alpha+beta-proteins (Cid et al., 1992)
Normalized hydrophobicity scales for alpha/beta-proteins (Cid et al., 1992)
Normalized average hydrophobicity scales (Cid et al., 1992)
Partial specific volume (Cohn-Edsall, 1943)
Normalized frequency of middle helix (Crawford et al., 1973)
Normalized frequency of beta-sheet (Crawford et al., 1973)
Normalized frequency of turn (Crawford et al., 1973)
Size (Dawson, 1972)
Amino acid composition (Dayhoff et al., 1978a)
Relative mutability (Dayhoff et al., 1978b)
Membrane preference for cytochrome b: MPH89 (Degli Esposti et al., 1990)
Average membrane preference: AMP07 (Degli Esposti et al., 1990)
Consensus normalized hydrophobicity scale (Eisenberg, 1984)
Solvation free energy (Eisenberg-McLachlan, 1986)
Atom-based hydrophobic moment (Eisenberg-McLachlan, 1986)
Direction of hydrophobic moment (Eisenberg-McLachlan, 1986)
Molecular weight (Fasman, 1976)
Melting point (Fasman, 1976)
Optical rotation (Fasman, 1976)
pK-N (Fasman, 1976)
pK-C (Fasman, 1976)
Hydrophobic parameter pi (Fauchere-Pliska, 1983)

Graph shape index (Fauchere et al., 1988)
Smoothed upsilon steric parameter (Fauchere et al., 1988)
Normalized van der Waals volume (Fauchere et al., 1988)
STERIMOL length of the side chain (Fauchere et al., 1988)
STERIMOL minimum width of the side chain (Fauchere et al., 1988)
STERIMOL maximum width of the side chain (Fauchere et al., 1988)
N.m.r. chemical shift of alpha-carbon (Fauchere et al., 1988)
Localized electrical effect (Fauchere et al., 1988)
Number of hydrogen bond donors (Fauchere et al., 1988)
Number of full nonbonding orbitals (Fauchere et al., 1988)
Positive charge (Fauchere et al., 1988)
Negative charge (Fauchere et al., 1988)
pK-a(RCOOH) (Fauchere et al., 1988)
Helix-coil equilibrium constant (Finkelstein-Ptitsyn, 1977)
Helix initiation parameter at posision i-1 (Finkelstein et al., 1991)
Helix initiation parameter at posision i,i+1,i+2 (Finkelstein et al., 1991)
Helix termination parameter at posision j-2,j-1,j (Finkelstein et al., 1991)
Helix termination parameter at posision j+1 (Finkelstein et al., 1991)
Partition coefficient (Garel et al., 1973)
Alpha-helix indices (Geisow-Roberts, 1980)
Alpha-helix indices for alpha-proteins (Geisow-Roberts, 1980)
Alpha-helix indices for beta-proteins (Geisow-Roberts, 1980)
Alpha-helix indices for alpha/beta-proteins (Geisow-Roberts, 1980)
Beta-strand indices (Geisow-Roberts, 1980)
Beta-strand indices for beta-proteins (Geisow-Roberts, 1980)
Beta-strand indices for alpha/beta-proteins (Geisow-Roberts, 1980)
Aperiodic indices (Geisow-Roberts, 1980)
Aperiodic indices for alpha-proteins (Geisow-Roberts, 1980)
Aperiodic indices for beta-proteins (Geisow-Roberts, 1980)
Aperiodic indices for alpha/beta-proteins (Geisow-Roberts, 1980)
Hydrophobicity factor (Goldsack-Chalifoux, 1973)
Residue volume (Goldsack-Chalifoux, 1973)
Composition (Grantham, 1974)
Polarity (Grantham, 1974)
Volume (Grantham, 1974)
Partition energy (Guy, 1985)
Hydration number (Hopfinger, 1971), Cited by Charton-Charton (1982)
Hydrophilicity value (Hopp-Woods, 1981)
Heat capacity (Hutchens, 1970)
Absolute entropy (Hutchens, 1970)
Entropy of formation (Hutchens, 1970)
Normalized relative frequency of alpha-helix (Isogai et al., 1980)
Normalized relative frequency of extended structure (Isogai et al., 1980)
Normalized relative frequency of bend (Isogai et al., 1980)
Normalized relative frequency of bend R (Isogai et al., 1980)
Normalized relative frequency of bend S (Isogai et al., 1980)
Normalized relative frequency of helix end (Isogai et al., 1980)
Normalized relative frequency of double bend (Isogai et al., 1980)

Normalized relative frequency of coil (Isogai et al., 1980)
Average accessible surface area (Janin et al., 1978)
Percentage of buried residues (Janin et al., 1978)
Percentage of exposed residues (Janin et al., 1978)
Ratio of buried and accessible molar fractions (Janin, 1979)
Transfer free energy (Janin, 1979)
Hydrophobicity (Jones, 1975)
pK (-COOH) (Jones, 1975)
Relative frequency of occurrence (Jones et al., 1992)
Relative mutability (Jones et al., 1992)
Amino acid distribution (Jukes et al., 1975)
Sequence frequency (Jungck, 1978)
Average relative probability of helix (Kanehisa-Tsong, 1980)
Average relative probability of beta-sheet (Kanehisa-Tsong, 1980)
Average relative probability of inner helix (Kanehisa-Tsong, 1980)
Average relative probability of inner beta-sheet (Kanehisa-Tsong, 1980)
Flexibility parameter for no rigid neighbors (Karplus-Schulz, 1985)
Flexibility parameter for one rigid neighbor (Karplus-Schulz, 1985)
Flexibility parameter for two rigid neighbors (Karplus-Schulz, 1985)
The Kerr-constant increments (Khanarian-Moore, 1980)
Net charge (Klein et al., 1984)
Side chain interaction parameter (Krigbaum-Rubin, 1971)
Side chain interaction parameter (Krigbaum-Komoriya, 1979)
Fraction of site occupied by water (Krigbaum-Komoriya, 1979)
Side chain volume (Krigbaum-Komoriya, 1979)
Hydropathy index (Kyte-Doolittle, 1982)
Transfer free energy, CHP/water (Lawson et al., 1984)
Hydrophobic parameter (Levitt, 1976)
Distance between C-alpha and centroid of side chain (Levitt, 1976)
Side chain angle theta(AAR) (Levitt, 1976)
Side chain torsion angle phi(AAAR) (Levitt, 1976)
Radius of gyration of side chain (Levitt, 1976)
van der Waals parameter R0 (Levitt, 1976)
van der Waals parameter epsilon (Levitt, 1976)
Normalized frequency of alpha-helix, with weights (Levitt, 1978)
Normalized frequency of beta-sheet, with weights (Levitt, 1978)
Normalized frequency of reverse turn, with weights (Levitt, 1978)
Normalized frequency of alpha-helix, unweighted (Levitt, 1978)
Normalized frequency of beta-sheet, unweighted (Levitt, 1978)
Normalized frequency of reverse turn, unweighted (Levitt, 1978)
Frequency of occurrence in beta-bends (Lewis et al., 1971)
Conformational preference for all beta-strands (Lifson-Sander, 1979)
Conformational preference for parallel beta-strands (Lifson-Sander, 1979)
Conformational preference for antiparallel beta-strands (Lifson-Sander, 1979)
Average surrounding hydrophobicity (Manavalan-Ponnuswamy, 1978)
Normalized frequency of alpha-helix (Maxfield-Scheraga, 1976)
Normalized frequency of extended structure (Maxfield-Scheraga, 1976)
Normalized frequency of zeta R (Maxfield-Scheraga, 1976)

Normalized frequency of left-handed alpha-helix (Maxfield-Scheraga, 1976)
Normalized frequency of zeta L (Maxfield-Scheraga, 1976)
Normalized frequency of alpha region (Maxfield-Scheraga, 1976)
Refractivity (McMeekin et al., 1964), Cited by Jones (1975)
Retention coefficient in HPLC, pH7.4 (Meek, 1980)
Retention coefficient in HPLC, pH2.1 (Meek, 1980)
Retention coefficient in NaClO4 (Meek-Rossetti, 1981)
Retention coefficient in NaH2PO4 (Meek-Rossetti, 1981)
Average reduced distance for C-alpha (Meirovitch et al., 1980)
Average reduced distance for side chain (Meirovitch et al., 1980)
Average side chain orientation angle (Meirovitch et al., 1980)
Effective partition energy (Miyazawa-Jernigan, 1985)
Normalized frequency of alpha-helix (Nagano, 1973)
Normalized frequency of bata-structure (Nagano, 1973)
Normalized frequency of coil (Nagano, 1973)
AA composition of total proteins (Nakashima et al., 1990)
SD of AA composition of total proteins (Nakashima et al., 1990)
AA composition of mt-proteins (Nakashima et al., 1990)
Normalized composition of mt-proteins (Nakashima et al., 1990)
AA composition of mt-proteins from animal (Nakashima et al., 1990)
Normalized composition from animal (Nakashima et al., 1990)
AA composition of mt-proteins from fungi and plant (Nakashima et al., 1990)
Normalized composition from fungi and plant (Nakashima et al., 1990)
AA composition of membrane proteins (Nakashima et al., 1990)
Normalized composition of membrane proteins (Nakashima et al., 1990)
Transmembrane regions of non-mt-proteins (Nakashima et al., 1990)
Transmembrane regions of mt-proteins (Nakashima et al., 1990)
Ratio of average and computed composition (Nakashima et al., 1990)
AA composition of CYT of single-spanning proteins (Nakashima-Nishikawa, 1992)
AA composition of CYT2 of single-spanning proteins (Nakashima-Nishikawa, 1992)
AA composition of EXT of single-spanning proteins (Nakashima-Nishikawa, 1992)
AA composition of EXT2 of single-spanning proteins (Nakashima-Nishikawa, 1992)
AA composition of MEM of single-spanning proteins (Nakashima-Nishikawa, 1992)
AA composition of CYT of multi-spanning proteins (Nakashima-Nishikawa, 1992)
AA composition of EXT of multi-spanning proteins (Nakashima-Nishikawa, 1992)
AA composition of MEM of multi-spanning proteins (Nakashima-Nishikawa, 1992)
8 A contact number (Nishikawa-Ooi, 1980)
14 A contact number (Nishikawa-Ooi, 1986)
Transfer energy, organic solvent/water (Nozaki-Tanford, 1971)
Average non-bonded energy per atom (Oobatake-Ooi, 1977)
Short and medium range non-bonded energy per atom (Oobatake-Ooi, 1977)
Long range non-bonded energy per atom (Oobatake-Ooi, 1977)
Average non-bonded energy per residue (Oobatake-Ooi, 1977)
Short and medium range non-bonded energy per residue (Oobatake-Ooi, 1977)
Optimized beta-structure-coil equilibrium constant (Oobatake et al., 1985)
Optimized propensity to form reverse turn (Oobatake et al., 1985)
Optimized transfer energy parameter (Oobatake et al., 1985)
Optimized average non-bonded energy per atom (Oobatake et al., 1985)

Optimized side chain interaction parameter (Oobatake et al., 1985)
Normalized frequency of alpha-helix from LG (Palau et al., 1981)
Normalized frequency of alpha-helix from CF (Palau et al., 1981)
Normalized frequency of beta-sheet from LG (Palau et al., 1981)
Normalized frequency of beta-sheet from CF (Palau et al., 1981)
Normalized frequency of turn from LG (Palau et al., 1981)
Normalized frequency of turn from CF (Palau et al., 1981)
Normalized frequency of alpha-helix in all-alpha class (Palau et al., 1981)
Normalized frequency of alpha-helix in alpha+beta class (Palau et al., 1981)
Normalized frequency of alpha-helix in alpha/beta class (Palau et al., 1981)
Normalized frequency of beta-sheet in all-beta class (Palau et al., 1981)
Normalized frequency of beta-sheet in alpha+beta class (Palau et al., 1981)
Normalized frequency of beta-sheet in alpha/beta class (Palau et al., 1981)
Normalized frequency of turn in all-alpha class (Palau et al., 1981)
Normalized frequency of turn in all-beta class (Palau et al., 1981)
Normalized frequency of turn in alpha+beta class (Palau et al., 1981)
Normalized frequency of turn in alpha/beta class (Palau et al., 1981)
HPLC parameter (Parker et al., 1986)
Partition coefficient (Pliska et al., 1981)
Surrounding hydrophobicity in folded form (Ponnuswamy et al., 1980)
Average gain in surrounding hydrophobicity (Ponnuswamy et al., 1980)
Average gain ratio in surrounding hydrophobicity (Ponnuswamy et al., 1980)
Surrounding hydrophobicity in alpha-helix (Ponnuswamy et al., 1980)
Surrounding hydrophobicity in beta-sheet (Ponnuswamy et al., 1980)
Surrounding hydrophobicity in turn (Ponnuswamy et al., 1980)
Accessibility reduction ratio (Ponnuswamy et al., 1980)
Average number of surrounding residues (Ponnuswamy et al., 1980)
Intercept in regression analysis (Prabhakaran-Ponnuswamy, 1982)
Slope in regression analysis x 1.0E1 (Prabhakaran-Ponnuswamy, 1982)
Correlation coefficient in regression analysis (Prabhakaran-Ponnuswamy, 1982)
Hydrophobicity (Prabhakaran, 1990)
Relative frequency in alpha-helix (Prabhakaran, 1990)
Relative frequency in beta-sheet (Prabhakaran, 1990)
Relative frequency in reverse-turn (Prabhakaran, 1990)
Helix-coil equilibrium constant (Ptitsyn-Finkelstein, 1983)
Beta-coil equilibrium constant (Ptitsyn-Finkelstein, 1983)
Weights for alpha-helix at the window position of -6 (Qian-Sejnowski, 1988)
Weights for alpha-helix at the window position of -5 (Qian-Sejnowski, 1988)
Weights for alpha-helix at the window position of -4 (Qian-Sejnowski, 1988)
Weights for alpha-helix at the window position of -3 (Qian-Sejnowski, 1988)
Weights for alpha-helix at the window position of -2 (Qian-Sejnowski, 1988)
Weights for alpha-helix at the window position of -1 (Qian-Sejnowski, 1988)
Weights for alpha-helix at the window position of 0 (Qian-Sejnowski, 1988)
Weights for alpha-helix at the window position of 1 (Qian-Sejnowski, 1988)
Weights for alpha-helix at the window position of 2 (Qian-Sejnowski, 1988)
Weights for alpha-helix at the window position of 3 (Qian-Sejnowski, 1988)
Weights for alpha-helix at the window position of 4 (Qian-Sejnowski, 1988)
Weights for alpha-helix at the window position of 5 (Qian-Sejnowski, 1988)

Weights for alpha-helix at the window position of 6 (Qian-Sejnowski, 1988)
Weights for beta-sheet at the window position of -6 (Qian-Sejnowski, 1988)
Weights for beta-sheet at the window position of -5 (Qian-Sejnowski, 1988)
Weights for beta-sheet at the window position of -4 (Qian-Sejnowski, 1988)
Weights for beta-sheet at the window position of -3 (Qian-Sejnowski, 1988)
Weights for beta-sheet at the window position of -2 (Qian-Sejnowski, 1988)
Weights for beta-sheet at the window position of -1 (Qian-Sejnowski, 1988)
Weights for beta-sheet at the window position of 0 (Qian-Sejnowski, 1988)
Weights for beta-sheet at the window position of 1 (Qian-Sejnowski, 1988)
Weights for beta-sheet at the window position of 2 (Qian-Sejnowski, 1988)
Weights for beta-sheet at the window position of 3 (Qian-Sejnowski, 1988)
Weights for beta-sheet at the window position of 4 (Qian-Sejnowski, 1988)
Weights for beta-sheet at the window position of 5 (Qian-Sejnowski, 1988)
Weights for beta-sheet at the window position of 6 (Qian-Sejnowski, 1988)
Weights for coil at the window position of -6 (Qian-Sejnowski, 1988)
Weights for coil at the window position of -5 (Qian-Sejnowski, 1988)
Weights for coil at the window position of -4 (Qian-Sejnowski, 1988)
Weights for coil at the window position of -3 (Qian-Sejnowski, 1988)
Weights for coil at the window position of -2 (Qian-Sejnowski, 1988)
Weights for coil at the window position of -1 (Qian-Sejnowski, 1988)
Weights for coil at the window position of 0 (Qian-Sejnowski, 1988)
Weights for coil at the window position of 1 (Qian-Sejnowski, 1988)
Weights for coil at the window position of 2 (Qian-Sejnowski, 1988)
Weights for coil at the window position of 3 (Qian-Sejnowski, 1988)
Weights for coil at the window position of 4 (Qian-Sejnowski, 1988)
Weights for coil at the window position of 5 (Qian-Sejnowski, 1988)
Weights for coil at the window position of 6 (Qian-Sejnowski, 1988)
Average reduced distance for C-alpha (Rackovsky-Scheraga, 1977)
Average reduced distance for side chain (Rackovsky-Scheraga, 1977)
Side chain orientational preference (Rackovsky-Scheraga, 1977)
Average relative fractional occurrence in A0(i) (Rackovsky-Scheraga, 1982)
Average relative fractional occurrence in AR(i) (Rackovsky-Scheraga, 1982)
Average relative fractional occurrence in AL(i) (Rackovsky-Scheraga, 1982)
Average relative fractional occurrence in EL(i) (Rackovsky-Scheraga, 1982)
Average relative fractional occurrence in E0(i) (Rackovsky-Scheraga, 1982)
Average relative fractional occurrence in ER(i) (Rackovsky-Scheraga, 1982)
Average relative fractional occurrence in A0(i-1) (Rackovsky-Scheraga, 1982)
Average relative fractional occurrence in AR(i-1) (Rackovsky-Scheraga, 1982)
Average relative fractional occurrence in AL(i-1) (Rackovsky-Scheraga, 1982)
Average relative fractional occurrence in EL(i-1) (Rackovsky-Scheraga, 1982)
Average relative fractional occurrence in E0(i-1) (Rackovsky-Scheraga, 1982)
Average relative fractional occurrence in ER(i-1) (Rackovsky-Scheraga, 1982)
Value of theta(i) (Rackovsky-Scheraga, 1982)
Value of theta(i-1) (Rackovsky-Scheraga, 1982)
Transfer free energy from chx to wat (Radzicka-Wolfenden, 1988)
Transfer free energy from oct to wat (Radzicka-Wolfenden, 1988)
Transfer free energy from vap to chx (Radzicka-Wolfenden, 1988)
Transfer free energy from chx to oct (Radzicka-Wolfenden, 1988)

Transfer free energy from vap to oct (Radzicka-Wolfenden, 1988)
Accessible surface area (Radzicka-Wolfenden, 1988)
Energy transfer from out to in(95 Mean polarity (Radzicka-Wolfenden, 1988)
Relative preference value at N" (Richardson-Richardson, 1988)
Relative preference value at N' (Richardson-Richardson, 1988)
Relative preference value at N-cap (Richardson-Richardson, 1988)
Relative preference value at N1 (Richardson-Richardson, 1988)
Relative preference value at N2 (Richardson-Richardson, 1988)
Relative preference value at N3 (Richardson-Richardson, 1988)
Relative preference value at N4 (Richardson-Richardson, 1988)
Relative preference value at N5 (Richardson-Richardson, 1988)
Relative preference value at Mid (Richardson-Richardson, 1988)
Relative preference value at C5 (Richardson-Richardson, 1988)
Relative preference value at C4 (Richardson-Richardson, 1988)
Relative preference value at C3 (Richardson-Richardson, 1988)
Relative preference value at C2 (Richardson-Richardson, 1988)
Relative preference value at C1 (Richardson-Richardson, 1988)
Relative preference value at C-cap (Richardson-Richardson, 1988)
Relative preference value at C' (Richardson-Richardson, 1988)
Relative preference value at C" (Richardson-Richardson, 1988)
Information measure for alpha-helix (Robson-Suzuki, 1976)
Information measure for N-terminal helix (Robson-Suzuki, 1976)
Information measure for middle helix (Robson-Suzuki, 1976)
Information measure for C-terminal helix (Robson-Suzuki, 1976)
Information measure for extended (Robson-Suzuki, 1976)
Information measure for pleated-sheet (Robson-Suzuki, 1976)
Information measure for extended without H-bond (Robson-Suzuki, 1976)
Information measure for turn (Robson-Suzuki, 1976)
Information measure for N-terminal turn (Robson-Suzuki, 1976)
Information measure for middle turn (Robson-Suzuki, 1976)
Information measure for C-terminal turn (Robson-Suzuki, 1976)
Information measure for coil (Robson-Suzuki, 1976)
Information measure for loop (Robson-Suzuki, 1976)
Hydration free energy (Robson-Osguthorpe, 1979)
Mean area buried on transfer (Rose et al., 1985)
Mean fractional area loss (Rose et al., 1985)
Side chain hydropathy, uncorrected for solvation (Roseman, 1988)
Side chain hydropathy, corrected for solvation (Roseman, 1988)
Loss of Side chain hydropathy by helix formation (Roseman, 1988)
Transfer free energy (Simon, 1976), Cited by Charton-Charton (1982)
Principal component I (Sneath, 1966)
Principal component II (Sneath, 1966)
Principal component III (Sneath, 1966)
Principal component IV (Sneath, 1966)
Zimm-Bragg parameter s at 20 C (Sueki et al., 1984)
Zimm-Bragg parameter sigma x 1.0E4 (Sueki et al., 1984)
Optimal matching hydrophobicity (Sweet-Eisenberg, 1983)
Normalized frequency of alpha-helix (Tanaka-Scheraga, 1977)

Normalized frequency of isolated helix (Tanaka-Scheraga, 1977)
Normalized frequency of extended structure (Tanaka-Scheraga, 1977)
Normalized frequency of chain reversal R (Tanaka-Scheraga, 1977)
Normalized frequency of chain reversal S (Tanaka-Scheraga, 1977)
Normalized frequency of chain reversal D (Tanaka-Scheraga, 1977)
Normalized frequency of left-handed helix (Tanaka-Scheraga, 1977)
Normalized frequency of zeta R (Tanaka-Scheraga, 1977)
Normalized frequency of coil (Tanaka-Scheraga, 1977)
Normalized frequency of chain reversal (Tanaka-Scheraga, 1977)
Relative population of conformational state A (Vasquez et al., 1983)
Relative population of conformational state C (Vasquez et al., 1983)
Relative population of conformational state E (Vasquez et al., 1983)
Electron-ion interaction potential (Veljkovic et al., 1985)
Bitterness (Venanzi, 1984)
Transfer free energy to lipophilic phase (von Heijne-Blomberg, 1979)
Average interactions per side chain atom (Warme-Morgan, 1978)
RF value in high salt chromatography (Weber-Lacey, 1978)
Propensity to be buried inside (Wertz-Scheraga, 1978)
Free energy change of epsilon(i) to epsilon(ex) (Wertz-Scheraga, 1978)
Free energy change of alpha(Ri) to alpha(Rh) (Wertz-Scheraga, 1978)
Free energy change of epsilon(i) to alpha(Rh) (Wertz-Scheraga, 1978)
Polar requirement (Woese, 1973)
Hydration potential (Wolfenden et al., 1981)
Principal property value z1 (Wold et al., 1987)
Principal property value z2 (Wold et al., 1987)
Principal property value z3 (Wold et al., 1987)
Unfolding Gibbs energy in water, pH7.0 (Yutani et al., 1987)
Unfolding Gibbs energy in water, pH9.0 (Yutani et al., 1987)
Activation Gibbs energy of unfolding, pH7.0 (Yutani et al., 1987)
Activation Gibbs energy of unfolding, pH9.0 (Yutani et al., 1987)
Dependence of partition coefficient on ionic strength (Zaslavsky et al., 1982)
Hydrophobicity (Zimmerman et al., 1968)
Bulkiness (Zimmerman et al., 1968)
Polarity (Zimmerman et al., 1968)
Isoelectric point (Zimmerman et al., 1968)
RF rank (Zimmerman et al., 1968)
Normalized positional residue frequency at helix termini N4'(Aurora-Rose, 1998)
Normalized positional residue frequency at helix termini N"' (Aurora-Rose, 1998)
Normalized positional residue frequency at helix termini N" (Aurora-Rose, 1998)
Normalized positional residue frequency at helix termini N'(Aurora-Rose, 1998)
Normalized positional residue frequency at helix termini Nc (Aurora-Rose, 1998)
Normalized positional residue frequency at helix termini N1 (Aurora-Rose, 1998)
Normalized positional residue frequency at helix termini N2 (Aurora-Rose, 1998)
Normalized positional residue frequency at helix termini N3 (Aurora-Rose, 1998)
Normalized positional residue frequency at helix termini N4 (Aurora-Rose, 1998)
Normalized positional residue frequency at helix termini N5 (Aurora-Rose, 1998)
Normalized positional residue frequency at helix termini C5 (Aurora-Rose, 1998)
Normalized positional residue frequency at helix termini C4 (Aurora-Rose, 1998)

Normalized positional residue frequency at helix termini C3 (Aurora-Rose, 1998)
Normalized positional residue frequency at helix termini C2 (Aurora-Rose, 1998)
Normalized positional residue frequency at helix termini C1 (Aurora-Rose, 1998)
Normalized positional residue frequency at helix termini Cc (Aurora-Rose, 1998)
Normalized positional residue frequency at helix termini C' (Aurora-Rose, 1998)
Normalized positional residue frequency at helix termini C" (Aurora-Rose, 1998)
Normalized positional residue frequency at helix termini C"' (Aurora-Rose, 1998)
Normalized positional residue frequency at helix termini C4' (Aurora-Rose, 1998)
Delta G values for the peptides extrapolated to 0 M urea (O'Neil-DeGrado, 1990)
Helix formation parameters (delta delta G) (O'Neil-DeGrado, 1990)
Normalized flexibility parameters (B-values), average (Vihinen et al., 1994)
Normalized flexibility parameters (B-values) for each residue surrounded by none rigid neighbours (Vihinen et al., 1994)
Normalized flexibility parameters (B-values) for each residue surrounded by one rigid neighbours (Vihinen et al., 1994)
Normalized flexibility parameters (B-values) for each residue surrounded by two rigid neighbours (Vihinen et al., 1994)
Free energy in alpha-helical conformation (Munoz-Serrano, 1994)
Free energy in alpha-helical region (Munoz-Serrano, 1994)
Free energy in beta-strand conformation (Munoz-Serrano, 1994)
Free energy in beta-strand region (Munoz-Serrano, 1994)
Free energy in beta-strand region (Munoz-Serrano, 1994)
Free energies of transfer of AcWl-X-LL peptides from bilayer interface to water (Wimley-White, 1996)
Thermodynamic beta sheet propensity (Kim-Berg, 1993)
Turn propensity scale for transmembrane helices (Monne et al., 1999)
Alpha helix propensity of position 44 in T4 lysozyme (Blaber et al., 1993)
p-Values of mesophilic proteins based on the distributions of B values (Parthasarathy-Murthy, 2000)
p-Values of thermophilic proteins based on the distributions of B values (Parthasarathy-Murthy, 2000)
Distribution of amino acid residues in the 18 non-redundant families of thermophilic proteins (Kumar et al., 2000)
Distribution of amino acid residues in the 18 non-redundant families of mesophilic proteins (Kumar et al., 2000)
Distribution of amino acid residues in the alpha-helices in thermophilic proteins (Kumar et al., 2000)
Distribution of amino acid residues in the alpha-helices in mesophilic proteins (Kumar et al., 2000)
Side-chain contribution to protein stability (kJ/mol) (Takano-Yutani, 2001)
Propensity of amino acids within pi-helices (Fodje-Al-Karadaghi, 2002)
Hydropathy scale based on self-information values in the two-state model (5 Hydropathy scale based on self-information values in the two-state model (9 Hydropathy scale based on self-information values in the two-state model (16 Hydropathy scale based on self-information values in the two-state model (20 Hydropathy scale based on self-information values in the two-state model (25 Hydropathy scale based on self-information values in the two-state model (36 Hydropathy scale based on self-information values in the two-state model (50 Averaged turn propensities in a transmembrane helix (Monne et al., 1999)
Alpha-helix propensity derived from designed sequences (Koehl-Levitt, 1999)
Beta-sheet propensity derived from designed sequences (Koehl-Levitt, 1999)

Composition of amino acids in extracellular proteins (percent) (Cedano et al., 1997)
Composition of amino acids in anchored proteins (percent) (Cedano et al., 1997)
Composition of amino acids in membrane proteins (percent) (Cedano et al., 1997)
Composition of amino acids in intracellular proteins (percent) (Cedano et al., 1997)
Composition of amino acids in nuclear proteins (percent) (Cedano et al., 1997)
Surface composition of amino acids in intracellular proteins of thermophiles (percent) (Fukuchi-Nishikawa, 2001)
Surface composition of amino acids in intracellular proteins of mesophiles (percent) (Fukuchi-Nishikawa, 2001)
Surface composition of amino acids in extracellular proteins of mesophiles (percent) (Fukuchi-Nishikawa, 2001)
Surface composition of amino acids in nuclear proteins (percent) (Fukuchi-Nishikawa, 2001)
Interior composition of amino acids in intracellular proteins of thermophiles (percent) (Fukuchi-Nishikawa, 2001)
Interior composition of amino acids in intracellular proteins of mesophiles (percent) (Fukuchi-Nishikawa, 2001)
Interior composition of amino acids in extracellular proteins of mesophiles (percent) (Fukuchi-Nishikawa, 2001)
Interior composition of amino acids in nuclear proteins (percent) (Fukuchi-Nishikawa, 2001)
Entire chain composition of amino acids in intracellular proteins of thermophiles (percent) (Fukuchi-Nishikawa, 2001)
Entire chain composition of amino acids in intracellular proteins of mesophiles (percent) (Fukuchi-Nishikawa, 2001)
Entire chain composition of amino acids in extracellular proteins of mesophiles (percent) (Fukuchi-Nishikawa, 2001)
Entire chain compositino of amino acids in nuclear proteins (percent) (Fukuchi-Nishikawa, 2001)
Screening coefficients gamma, local (Avbelj, 2000)
Screening coefficients gamma, non-local (Avbelj, 2000)
Slopes tripeptide, FDPB VFF neutral (Avbelj, 2000)
Slopes tripeptides, LD VFF neutral (Avbelj, 2000)
Slopes tripeptide, FDPB VFF noside (Avbelj, 2000)
Slopes tripeptide FDPB VFF all (Avbelj, 2000)
Slopes tripeptide FDPB PARSE neutral (Avbelj, 2000)
Slopes dekapeptide, FDPB VFF neutral (Avbelj, 2000)
Slopes proteins, FDPB VFF neutral (Avbelj, 2000)
Side-chain conformation by gaussian evolutionary method (Yang et al., 2002)
Amphiphilicity index (Mitaku et al., 2002)
Volumes including the crystallographic waters using the ProtOr (Tsai et al., 1999)
Volumes not including the crystallographic waters using the ProtOr (Tsai et al., 1999)
Electron-ion interaction potential values (Cosic, 1994)
Hydrophobicity scales (Ponnuswamy, 1993)
Hydrophobicity coefficient in RP-HPLC, C18 with 0.1 Hydrophobicity coefficient in RP-HPLC, C8 with 0.1 Hydrophobicity coefficient in RP-HPLC, C4 with 0.1 Hydrophobicity coefficient in RP-HPLC, C18 with 0.1 Hydrophilicity scale (Kuhn et al., 1995)
Retention coefficient at pH 2 (Guo et al., 1986)
Modified Kyte-Doolittle hydrophobicity scale (Juretic et al., 1998)
Interactivity scale obtained from the contact matrix (Bastolla et al., 2005)
Interactivity scale obtained by maximizing the mean of correlation coefficient over single-domain

globular proteins (Bastolla et al., 2005)
Interactivity scale obtained by maximizing the mean of correlation coefficient over pairs of sequences sharing the TIM barrel fold (Bastolla et al., 2005)
Linker propensity index (Suyama-Ohara, 2003)
Knowledge-based membrane-propensity scale from 1D_Helix in MPtopo databases (Punta-Maritan, 2003)
Knowledge-based membrane-propensity scale from 3D_Helix in MPtopo databases (Punta-Maritan, 2003)
Linker propensity from all dataset (George-Heringa, 2003)
Linker propensity from 1-linker dataset (George-Heringa, 2003)
Linker propensity from 2-linker dataset (George-Heringa, 2003)
Linker propensity from 3-linker dataset (George-Heringa, 2003)
Linker propensity from small dataset (linker length is less than six residues) (George-Heringa, 2003)
Linker propensity from medium dataset (linker length is between six and 14 residues) (George-Heringa, 2003)
Linker propensity from long dataset (linker length is greater than 14 residues) (George-Heringa, 2003)
Linker propensity from helical (annotated by DSSP) dataset (George-Heringa, 2003)
Linker propensity from non-helical (annotated by DSSP) dataset (George-Heringa, 2003)
The stability scale from the knowledge-based atom-atom potential (Zhou-Zhou, 2004)
The relative stability scale extracted from mutation experiments (Zhou-Zhou, 2004)
Buriability (Zhou-Zhou, 2004)
Linker index (Bae et al., 2005)
Mean volumes of residues buried in protein interiors (Harpaz et al., 1994)
Average volumes of residues (Pontius et al., 1996)
Hydrostatic pressure asymmetry index, PAI (Di Giulio, 2005)
Hydrophobicity index (Wolfenden et al., 1979)
Average internal preferences (Olsen, 1980)
Hydrophobicity-related index (Kidera et al., 1985)
Apparent partition energies calculated from Wertz-Scheraga index (Guy, 1985)
Apparent partition energies calculated from Robson-Osguthorpe index (Guy, 1985)
Apparent partition energies calculated from Janin index (Guy, 1985)
Apparent partition energies calculated from Chothia index (Guy, 1985)
Hydropathies of amino acid side chains, neutral form (Roseman, 1988)
Hydropathies of amino acid side chains, pi-values in pH 7.0 (Roseman, 1988)
Weights from the IFH scale (Jacobs-White, 1989)
Hydrophobicity index, 3.0 pH (Cowan-Whittaker, 1990)
Scaled side chain hydrophobicity values (Black-Mould, 1991)
Hydrophobicity scale from native protein structures (Casari-Sippl, 1992)
NNEIG index (Cornette et al., 1987)
SWEIG index (Cornette et al., 1987)
PRIFT index (Cornette et al., 1987)
PRILS index (Cornette et al., 1987)
ALTFT index (Cornette et al., 1987)
ALTLS index (Cornette et al., 1987)
TOTFT index (Cornette et al., 1987)
TOTLS index (Cornette et al., 1987)
Relative partition energies derived by the Bethe approximation (Miyazawa-Jernigan, 1999)

Optimized relative partition energies - method A (Miyazawa-Jernigan, 1999)
Optimized relative partition energies - method B (Miyazawa-Jernigan, 1999)
Optimized relative partition energies - method C (Miyazawa-Jernigan, 1999)
Optimized relative partition energies - method D (Miyazawa-Jernigan, 1999)
Hydrophobicity index (Engelman et al., 1986)
Hydrophobicity index (Fasman, 1989)

Source

http://www.genome.jp/aaindex

References

From the original aaindex documentation:

Please cite the following references when making use of the database:

Kawashima, S. and Kanehisa, M. (2000) AAindex: amino acid index database. *Nucleic Acids Res.*, **28**:374.

Tomii, K. and Kanehisa, M. (1996) Analysis of amino acid indices and mutation matrices for sequence comparison and structure prediction of proteins. *Protein Eng.*, **9**:27-36.

Nakai, K., Kidera, A., and Kanehisa, M. (1988) Cluster analysis of amino acid indices for prediction of protein structure and function. *Protein Eng.* **2**:93-100.

Examples

```
#
# Load data:
#

data(aaindex)

#
# Supose that we need the Kyte & Doolittle Hydrophaty index. We first look
# at the entries with Kyte as author:
#

which(sapply(aaindex, function(x) length(grep("Kyte", x$A)) != 0))

#
# This should return that entry number 151 named KYTJ820101 is the only
# one that fit our request. We can access to it by position or by name,
# for instance:
#

aaindex[[151]]$I
aaindex[["KYTJ820101"]]$I
aaindex$KYTJ820101$I
```

acnucopen *open and close a remote access to an ACNUC database*

Description

These are low level functions to start and stop a remote access to an ACNUC database.

Usage

```
acnucopen(db, socket, challenge = NA)
acnucclose(socket)
clientid(id = paste("seqinr_", packageDescription("seqinr")$Version, sep = ""), soc
quitacnuc(socket)
```

Arguments

db	the remote ACNUC database name
socket	an object of class sockconn connecting to an ACNUC server
challenge	unimplemented yet
id	client ID definition defaulting to seqinr + package version number
verbose	logical, if TRUE mode verbose is on

Details

these low level functions are usually not used directly by the user. Use choosebank to open a remote ACNUC database and closebank to close it.

Value

For openacnuc a list with the following components: type : the type of database that was opened. totseqs, totspec, totkey : total number of seqs, species, keywords in opened database. ACC_LENGTH, L_MNEMO, WIDTH_KW, WIDTH_SP, WIDTH_SMJ, WIDTH_AUT, WIDTH_BIB, lrtxt, SUBINLNG: max lengths of record keys in database.

Author(s)

J.R. Lobry

References

```
citation("seqinr")
```

See Also

choosebank, closebank

Examples

```
## Not run: # Need internet connection
 mysocket <- socketConnection( host = "pbil.univ-lyon1.fr",
   port = 5558, server = FALSE, blocking = TRUE)
 readLines(mysocket, n = 1) # OK acnuc socket started
 acnucopen("emblTP", socket = mysocket) -> res
 expected <- c("EMBL", "14138095", "236401", "1186228", "8",
   "16", "40", "40", "20", "20", "40", "60", "63")
 stopifnot(all(unlist(res) == expected))
 tryalreadyopen <- try(acnucopen("emblTP", socket = mysocket))
 stopifnot(inherits(tryalreadyopen, "try-error"))
 # Need a fresh socket because acnucopen() close it if error:
 mysocket <- socketConnection( host = "pbil.univ-lyon1.fr",
   port = 5558, server = FALSE, blocking = TRUE)
 tryoff <-  try(acnucopen("off", socket = mysocket))
 stopifnot(inherits(tryoff, "try-error"))

 mysocket <- socketConnection( host = "pbil.univ-lyon1.fr",
   port = 5558, server = FALSE, blocking = TRUE)
 tryinexistent <-  try(acnucopen("tagadatagadatsointsoin", socket = mysocket))
 stopifnot(inherits(tryinexistent, "try-error"))

 mysocket <- socketConnection( host = "pbil.univ-lyon1.fr",
   port = 5558, server = FALSE, blocking = TRUE)
 trycloseunopened <- try(acnucclose(mysocket))
 stopifnot(inherits(trycloseunopened, "try-error"))

## End(Not run)
```

`alllistranks` *To get the count of existing lists and all their ranks on server*

Description

This is a low level function to get the total number of list and all their ranks in an opened database.

Usage

```
alllistranks(socket = autosocket(), verbose = FALSE)
alr(socket = autosocket(), verbose = FALSE)
```

Arguments

`socket`	an object of class `sockconn` connecting to a remote ACNUC database (default is a socket to the last opened database).
`verbose`	if `TRUE`, verbose mode is on

Details

This low level function is usually not used directly by the user.

Value

A list with two components:

`count`	count of existing lists
`rank`	their rank

Author(s)

J.R. Lobry

References

```
citation("seqinr")
```

See Also

choosebank, query

Examples

```
 ## Not run:
# Need internet connection
 choosebank("emblTP")
 query("tmp1", "sp=Borrelia burgdorferi", virtual = TRUE)
 query("tmp2", "sp=Borrelia burgdorferi", virtual = TRUE)
 query("tmp3", "sp=Borrelia burgdorferi", virtual = TRUE)
 (result <- alllistranks())
 stopifnot(result$count == 3)   # Three ACNUC lists
 stopifnot(result$ranks == 2:4) # Starting at rank 2
 #
 # Summay of current lists defined on the ACNUC server:
 #
 sapply(result$ranks, getliststate)
 closebank()

## End(Not run)
```

`amb` *Expansion of IUPAC nucleotide symbols*

Description

This function returns the list of nucleotide matching a given IUPAC nucleotide symbol, for instance `c("c", "g")` for `"s"`.

Usage

```
amb(base, forceToLower = TRUE, checkBase = TRUE,
IUPAC = s2c("acgturymkswbdhvn"), u2t = TRUE)
```

Arguments

base	an IUPAC symbol for a nucleotide as a single character
forceToLower	if TRUE the base is forced to lower case
checkBase	if TRUE the character is checked to belong to the allowed IUPAC symbol list
IUPAC	the list of allowed IUPAC symbols
u2t	if TRUE "u" for uracil in RNA are changed into "t" for thymine in DNA

Details

Non ambiguous bases are returned unchanged (except for "u" when u2t is TRUE).

Value

When base is missing, the list of IUPAC symbols is returned, otherwise a vector with expanded symbols.

Author(s)

J.R. Lobry

References

The nomenclature for incompletely specified bases in nucleic acid sequences at: http://www.chem.qmul.ac.uk/iubmb/misc/naseq.html

```
citation("seqinr")
```

See Also

See bma for the reverse operation. Use tolower to change upper case letters into lower case letters.

Examples

```
#
# The list of IUPAC symbols:
#

amb()

#
# And their expansion:
#

sapply(amb(), amb)
```

as.matrix.alignment *as.matrix.alignment*

Description

Converts an alignment into a matrix of characters

Usage

```
## S3 method for class 'alignment':
as.matrix(x, ...)
```

Arguments

x	an object of the class alignment.
...	additional arguments to be passed to or from methods.

Value

A matrix of characters.

Author(s)

J.R. Lobry

See Also

read.alignment

Examples

```
phylip <- read.alignment(file = system.file("sequences/test.phylip", package = "seqinr"),
as.matrix(phylip)
```

autosocket *Returns a socket to the last opened database*

Description

This is a low level function that is mainly used to select automatically the last opened ACNUC database for functions using sockets.

Usage

```
autosocket()
```

Value

An object of class sockconn.

Author(s)

J.R. Lobry

References

http://pbil.univ-lyon1.fr/databases/acnuc.html

```
citation("seqinr")
```

See Also

choosebank, connections.

Examples

```
## Not run:
 #Need internet connection
  choosebank("emblTP")
  autosocket()
  closebank()

## End(Not run)
```

---

bma *Computing an IUPAC nucleotide symbol*

Description

This function returns the IUPAC symbol for a nucleotide sequence, for instance `c("c", "c", "g")` is coded by `"s"`.

Usage

```
bma(nucl, warn.non.IUPAC = TRUE, type = c("DNA", "RNA"))
```

Arguments

`nucl`	a nucleotide sequence as a vector of single chars
`warn.non.IUPAC`	if TRUE warns when no IUPAC symbol is possible
`type`	whether this is a DNA or a RNA sequence

Details

The sequence is forced in lower case letters and ambiguous bases are expanded before trying to find an IUPAC symbol.

Value

A single IUPAC symbol in lower case, or NA when this is not possible.

Author(s)

J.R. Lobry

References

The nomenclature for incompletely specified bases in nucleic acid sequences at: http://www.chem.qmul.ac.uk/iubmb/misc/naseq.html

```
citation("seqinr")
```

See Also

See amb for the reverse operation. Use toupper to change lower case letters into upper case letters.

Examples

```
stopifnot(bma(s2c("atatattttata")) == "w")
stopifnot(bma(s2c("gcggcgcgcggc")) == "s")
stopifnot(bma(s2c("ACGT")) == "n")
stopifnot(is.na(bma(s2c("atatttt---tatat")))) # a warning is issued
```

---

c2s *conversion of a vector of chars into a string*

Description

This is a simple utility function to convert a vector of chars such as c("m", "e", "r", "g", "e", "d") into a single string such as "merged".

Usage

```
c2s(chars = c("m", "e", "r", "g", "e", "d"))
```

Arguments

chars	a vector of chars

Value

a string

Author(s)

J.R. Lobry

References

```
citation("seqinr")
```

See Also

s2c

Examples

```
c2s( c("m","e","r","g","e","d") )
```

---

cai *Codon Adaptation Index*

Description

The Codon Adaptation Index (Sharp and Li 1987) is the most popular index of gene expressivity with about 1000 citations 20 years after its publication. Its values range from 0 (low) to 1 (high). The implementation here is intended to work exactly as in the program codonW written by by John Peden during his PhD thesis under the supervision of P.M. Sharp.

Usage

```
cai(seq, w, numcode = 1, zero.threshold = 0.0001, zero.to = 0.01)
```

Arguments

seq	a coding sequence as a vector of single characters
w	a vector for the relative adaptiveness of each codon
numcode	the genetic code number as in translate
zero.threshold	a value in w below this threshold is considered as zero
zero.to	a value considered as zero in w is forced to this value. The default is from Bulmer (1988).

Details

Adapted from the documentation of the CAI function in the program codonW writen by John Peden: CAI is a measurement of the relative adaptiveness of the codon usage of a gene towards the codon usage of highly expressed genes. The relative adaptiveness (w) of each codon is the ratio of the usage of each codon, to that of the most abundant codon for the same amino acid. The CAI index is defined as the geometric mean of these relative adaptiveness values. Non-synonymous codons and termination codons (genetic code dependent) are excluded. To aid computation, the CAI is calculated as using a natural log summation, To prevent a codon having a relative adaptiveness value of zero, which could result in a CAI of zero; these codons have fitness of zero (<.0001) are adjusted to 0.01.

Value

A single numerical value for the CAI.

Author(s)

J.R. Lobry

References

Sharp, P.M., Li, W.-H. (1987) The codon adaptation index - a measure of directional synonymous codon usage bias, and its potential applications. *Nucleic Acids Research*, **15**:1281-1295.

Bulmer, M. (1988). Are codon usage patterns in unicellular organisms determined by selection-mutation balance. *Journal of Evolutionary Biology*, **1**:15-26.

Peden, J.F. (1999) Analysis of codon usage. PhD Thesis, University of Nottingham, UK.

The program codonW used here for comparison is available at http://codonw.sourceforge.net/ under a GPL licence.

citation("seqinr").

See Also

caitab for some w values from codonW. uco for codon usage tabulation.

Examples

```
#
# How to reproduce the results obtained with the C program codonW
# version 1.4.4 writen by John Peden. We use here the "input.dat"
# test file from codonW (Saccharomyces cerevisiae).
#
  inputdatfile <- system.file("sequences/input.dat", package = "seqinr")
  input <- read.fasta(file = inputdatfile) # read the FASTA file
#
# Import results obtained with codonW
#
  scucofile <- system.file("sequences/scuco.txt", package = "seqinr")
  scuco.res <- read.table(scucofile, header = TRUE) # read codonW result file
#
```

```
# Use w for Saccharomyces cerevisiae
#
  data(caitab)
  w <- caitab$sc
#
# Compute CAI and compare results:
#
  cai.res <- sapply(input, cai, w = w)
  plot(cai.res, scuco.res$CAI,
    main = "Comparison of seqinR and codonW results",
    xlab = "CAI from seqinR",
    ylab = "CAI from codonW",
    las = 1)
  abline(c(0,1))
```

`caitab` *Codon Adaptation Index (CAI) w tables*

Description

Information about a preferred set of codons for highly expressed genes in three species.

Usage

```
data(caitab)
```

Format

A data frame with 64 rows for the codons and the following 3 columns:

ec *Escherichia coli*

bs *Bacillus subtilis*

sc *Saccharomyces cerevisiae*

Details

Codons are given by `row.names(caitab)`.

Source

The data were hard-encoded in the C program codonW version 1.4.4 writen by John Peden available at http://codonw.sourceforge.net/. The data are from the file `codonW.h`. According to this source file, there were no reference for *Escherichia coli* and *Bacillus subtilis* and the reference for *Saccharomyces cerevisiae* was Sharp and Cowe (1991).

It turns out that the data for *Escherichia coli* and *Saccharomyces cerevisiae* are identical to table 1 in Sharp and Li (1987) where the missing values for the stop codons are represented here by zeros. All codons were documented by at least one count in both datasets.

The data for *Bacillus subtilis* are from table 2 in Shields and Sharp (1987). Missing values for stops codons are represented as previously by zeros, missing values for single-box amino-acids are represented by 1 here. Note that some codons were undocumented in this dataset and that a 0.5 value in absolute frequencies was already forced to avoid zeros. It is therefore impossible to use directly these data to obtain the exact expected CAI values as documented in `cai` because of overlapping with documented codons.

References

Sharp, P.M., Li, W.-H. (1987) The codon adaptation index - a measure of directional synonymous codon usage bias, and its potential applications. *Nucleic Acids Research*, **15**:1281-1295.

Shields, D.C., Sharp, P.M. (1987) Synonymous codon usage in *Bacillus subtilis* reflects both traditional selection and mutational biases. *Nucleic Acids Research*, **15**:8023-8040.

Sharp, P. M., Cowe, E. (1991). Synonymous codon usage in *Saccharomyces cerevisiae*. *Yeast*, **7**:657-678.

Peden, J.F. (1999) Analysis of codon usage. PhD Thesis, University of Nottingham, UK.

```
citation("seqinr")
```

See Also

`cai` for an example using this dataset to compute CAI values.

Examples

```
data(caitab)
```

`chargaff` *Base composition in ssDNA for 7 bacterial DNA*

Description

Long before the genomic era, it was possible to get some data for the global composition of single-stranded DNA chromosomes by direct chemical analyses. These data are from Chargaff's lab and give the base composition of the L (Ligth) strand for 7 bacterial chromosomes.

Usage

```
data(chargaff)
```

Format

A data frame with 7 observations on the following 4 variables.

[A] frequencies of A bases in percent

[G] frequencies of G bases in percent

[C] frequencies of C bases in percent

[T] frequencies of T bases in percent

Details

Data are from Table 2 in Rudner *et al.* (1969) for the L-strand. Data for *Bacillus subtilis* were taken from a previous paper: Rudner *et al.* (1968). This is in fact the average value observed for two different strains of *B. subtilis*: strain W23 and strain Mu8u5u16.
Denaturated chromosomes can be separated by a technique of intermitent gradient elution from a column of methylated albumin kieselguhr (MAK), into two fractions, designated, by virtue of their buoyant densities, as L (light) and H (heavy). The fractions can be hydrolyzed and subjected to chromatography to determined their global base composition.
The surprising result is that we have almost exactly A=T and C=G in single stranded-DNAs. The second paragraph page 157 in Rudner *et al.* (1969) says: "Our previous work on the complementary strands of *B. subtilis* DNA suggested an additional, entirely unexpected regularity, namely, the equality in either strand of 6-amino and 6-keto nucleotides (A + C = G + T). This relationship, which would normally have been regarded merely as the consequence of base-pairing in DNA duplex and would not have been predicted as a likely property of a single strand, is shown here to apply to all strand specimens isolated from denaturated DNA of the AT type (Table 2, preps. 1-4). It cannot yet be said to be established for the DNA specimens from the equimolar and GC types (nos. 5-7)."

Source

Rudner, R., Karkas, J.D., Chargaff, E. (1968) Separation of *B. subtilis* DNA into complementary strands, III. Direct Analysis. *Proceedings of the National Academy of Sciences of the United States of America*, **60**:921-922.
Rudner, R., Karkas, J.D., Chargaff, E. (1969) Separation of microbial deoxyribonucleic acids into complementary strands. *Proceedings of the National Academy of Sciences of the United States of America*, **63**:152-159.

References

Try `example(chargaff)` to mimic figure page 17 in http://pbil.univ-lyon1.fr/members/lobry/articles/HDR.pdf. The red areas correspond to non-allowed values beause the sum of the four bases frequencies cannot exceed 100%. The white areas correspond to possible values (more exactly to the projection from `R^4` to the corresponding `R^2` planes of the region of allowed values). The blue lines correspond to the very small subset of allowed values for which we have in addition PR2 state, that is `[A]=[T]` and `[C]=[G]`. Remember, these data are for ssDNA !

```
citation("seqinr")
```

Examples

```
data(chargaff)
op <- par(no.readonly = TRUE)
par(mfrow = c(4,4), mai = rep(0,4), xaxs = "i", yaxs = "i")
xlim <- ylim <- c(0, 100)

for( i in 1:4 )
{
  for( j in 1:4 )
  {
```

```
      if( i == j )
      {
        plot(chargaff[,i], chargaff[,j],t = "n", xlim = xlim, ylim = ylim,
        xlab = "", ylab = "", xaxt = "n", yaxt = "n")
        polygon(x = c(0, 0, 100, 100), y = c(0, 100, 100, 0), col = "lightgrey")
        for( k in seq(from = 0, to = 100, by = 10) )
        {
          lseg <- 3
          segments(k, 0, k, lseg)
          segments(k, 100 - lseg, k, 100)
          segments(0, k, lseg, k)
          segments(100 - lseg, k, 100, k)
        }
        string <- paste(names(chargaff)[i],"\n\n",xlim[1],"% -",xlim[2],"%")
        text(x=mean(xlim),y=mean(ylim), string, cex = 1.5)
      }
      else
      {
        plot(chargaff[,i], chargaff[,j], pch = 1, xlim = xlim, ylim = ylim,
        xlab = "", ylab = "", xaxt = "n", yaxt = "n", cex = 2)
        iname <- names(chargaff)[i]
        jname <- names(chargaff)[j]
        direct <- function() segments(0, 0, 50, 50, col="blue")
        invers <- function() segments(0, 50, 50, 0, col="blue")
        PR2 <- function()
        {
          if( iname == "[A]" & jname == "[T]" ) { direct(); return() }
          if( iname == "[T]" & jname == "[A]" ) { direct(); return() }
          if( iname == "[C]" & jname == "[G]" ) { direct(); return() }
          if( iname == "[G]" & jname == "[C]" ) { direct(); return() }
          invers()
        }
        PR2()
        polygon(x = c(0, 100, 100), y = c(100, 100, 0), col = "pink4")
        polygon(x = c(0, 0, 100), y = c(0, 100, 0))
      }
    }
  }
  # Clean up
  par(op)
```

`choosebank` *To select a database structured under ACNUC and located on the web*

Description

This function allows to select one of the databases structured under ACNUC and located on the web. Called without arguments, `choosebank()`, will return the list of available databases. Then, you can use `query` to make your query and get a list of sequence names. Remote access to ACNUC databases works by opening a socket connection on a port (for example on port number 5558 at

pbil.univ-lyon1.fr) and by communicating on this socket following the protocol described in the section `references`.

Usage

```
choosebank(bank = NA, host = "pbil.univ-lyon1.fr", port = 5558, server = FALSE,
                    blocking = TRUE, open = "a+", encoding = "", verbose = FALSE,
                    timeout = 5, infobank = FALSE, tagbank = NA)
```

Arguments

`bank`	string. The name of the bank. If NA, `choosebank` will return the names of all database known by the server.
`host`	string. Host name for port (see `socketConnection`)
`port`	integer. The TCP port number (see `socketConnection`)
`server`	logical. Should the socket be a client or a server? (see `socketConnection`)
`blocking`	logical. (see `socketConnection`)
`open`	string. A description of how to open the connection (see `socketConnection`)
`encoding`	string. The name of the encoding to be used. (see `socketConnection`)
`verbose`	logical. If TRUE, verbose mode is on
`timeout`	integer. The timeout in seconds for `socketConnection`. Default 5 seconds.
`infobank`	logical. If `infobank` is TRUE and `bank` is `NA`, a data.frame with all database informations will be returned
`tagbank`	string. If `bank` is `NA` and `tagbank` is documented, the names of special purposes databases are returned. Current allowed values are TP for frozen databases (TP is an acronym for "travaux pratiques" which means practicals in french, these databases are useful mainly for teaching so as to have stable results), TEST for test databases, and DEV for databases under development (unstable).

Details

When called without arguments, `choosebank()` returns a list of all the databases names known by the server, as a vector of string. When called with `choosebank(infobank = TRUE)`, a data.frame with more information is returned.

Value

When called with a regular bank name, an (invisible) list with 6 components:

`socket`	an object of class `socket`
`bankname`	the name of the bank
`banktype`	the type of the bank (GENBANK, EMBL, SWISSPROT, NBRF)
`totseqs`	the total number of sequences present in the opened database
`totspecs`	the total number of species present in the opened database
`totkeys`	the total number of keywords present in the opened database

	A vector of all available bank names.
bank	The name of the bank.
status	The bank status (on/of).
info	Short description of bank with last release date.

Note

The invisible list returned when a database is opened is stored in the variable banknameSocket in the global environment.

Author(s)

D. Charif, J.R. Lobry

References

For more information about the socket communication protocol with ACNUC please get at http://pbil.univ-lyon1.fr/databases/acnuc/remote_acnuc.html. To get the release date and content of all the databases located at the pbil, please look at the following url: http://pbil.univ-lyon1.fr/search/releases.php

Gouy, M., Milleret, F., Mugnier, C., Jacobzone, M., Gautier,C. (1984) ACNUC: a nucleic acid sequence data base and analysis system. *Nucl. Acids Res.*, **12**:121-127.
Gouy, M., Gautier, C., Attimonelli, M., Lanave, C., Di Paola, G. (1985) ACNUC - a portable retrieval system for nucleic acid sequence databases: logical and physical designs and usage. *Comput. Appl. Biosci.*, **3**:167-172.
Gouy, M., Gautier, C., Milleret, F. (1985) System analysis and nucleic acid sequence banks. *Biochimie*, **67**:433-436.

```
citation("seqinr")
```

See Also

query, connection, socketConnection

Examples

```
  ## Not run:
# Need internet connection
  # Show available databases:
  choosebank()
  # Show frozen databases:
  choosebank(tag = "TP")
  # Select a database:
  choosebank("emblTP", tag = "TP")
  # Do something with the database:
  myseq <- gfrag("LMFLCHR36", start = 1, length = 30)
  stopifnot(myseq == "cgcgtgctggcggcaatgaagcgttcgatg")
  # Close the database:
  closebank()
## End(Not run)
```

`closebank` *To close a remote ACNUC database*

Description

This function tries to close a remote ACNUC database.

Usage

```
closebank(socket = autosocket(), verbose = FALSE)
```

Arguments

`socket`	an object of class `sockconn` connecting to a remote ACNUC database (default is a socket to the last opened database).
`verbose`	Logical. If TRUE, verbose mode is on

Author(s)

J.R. Lobry

References

```
citation("seqinr")
```

See Also

`choosebank`

Examples

```
  ## Not run:
# Need internet connection
   choosebank("emblTP")
   closebank()

## End(Not run)
```

comp *complements a nucleic acid sequence*

Description

Complements a sequence, for instance if the sequence is "a","c","g","t" it returns "t","g","c","a". This is not the reverse complementary strand. This function can handle ambiguous bases if required.

Usage

```
comp(seq, forceToLower = TRUE, ambiguous = FALSE)
```

Arguments

seq	a DNA sequence as a vector of single chars
forceToLower	if TRUE character in seq are forced to lower case
ambiguous	if TRUE ambiguous bases in seq are handled

Value

a vector of characters which is the complement of the sequence, not the reverse complementary strand. Undefined values are returned as NA.

Author(s)

D. Charif, J.R. Lobry

References

```
citation("seqinr")
```

See Also

Because ssDNA sequences are always written in the 5'->3' direction, use rev(comp(seq)) to get the reverse complementary strand (see rev).

Examples

```
##
## Show that comp() does not return the reverve complementary strand:
##
c2s(comp(s2c("aaaattttggggcccc")))
##
## Show how to get the reverse complementary strand:
##
c2s(rev(comp(s2c("aaaattttggggcccc"))))
##
## Show what happens with non allowed values:
```

```
##
c2s(rev(comp(s2c("aaaaXttttYggggZcccc"))))
##
## Show what happens with ambiguous bases:
##
allbases <- s2c("abcdghkmstvwn")
comp(allbases) # NA are produced
comp(allbases, ambiguous = TRUE) # No more NA
##
## Routine sanity check:
##
stopifnot(identical(comp(allbases, ambiguous = TRUE), s2c("tvghcdmksabwn")))
```

`computePI` *To Compute the Theoretical Isoelectric Point*

Description

This function calculates the theoretical isoelectric point of a protein. Isoelectric point is the pH at which the protein has a neutral charge. This estimate does not account for the post-translational modifications.

Usage

```
computePI(seq)
```

Arguments

`seq`	Protein sequence as a vector of single chars in upper case

Value

The theoretical isoelectric point (pI) as a numerical vector of length one.

Note

Protein pI is calculated using pK values of amino acids described in Bjellqvist et al. See also `SEQINR.UTIL` for more details.

Author(s)

D. Charif and J.R. Lobry

References

The algorithm is the same as the one which is implemented at the following url: `http://www.expasy.org/tools/pi_tool-doc.html` but with many trials in case of convergence failure of the non-linear regression procedure. `citation("seqinr")`

See Also

SEQINR.UTIL

Examples

```
#
# Simple sanity check with all 20 amino-acids in one-letter code alphabetical order:
#
prot <- s2c("ACDEFGHIKLMNPQRSTVWY")
stopifnot(all.equal(computePI(prot), 6.78454))
#
# Read a protein sequence in a FASTA file and then compute its pI :
#
myProts <- read.fasta(file = system.file("sequences/seqAA.fasta", package = "seqinr"), seqty
computePI(myProts[[1]]) # Should be 8.534902
```

`consensus` *Consensus and profiles for sequence alignments*

Description

This function returns a consensus using variuous methods (see details) or a profile from a sequence alignment.

Usage

```
consensus(matali, method = c( "majority", "threshold", "IUPAC", "profile"),
  threshold = 0.60, warn.non.IUPAC = FALSE, type = c("DNA", "RNA"))
con(matali, method = c( "majority", "threshold", "IUPAC", "profile"),
  threshold = 0.60, warn.non.IUPAC = FALSE, type = c("DNA", "RNA"))
```

Arguments

`matali`	an object of class `alignment` as returned by `read.alignment`, or a matrix of characters.
`method`	select the method to use, see details.
`threshold`	for the `threshold` method, a numeric value beteen 0 and 1 indicating the minimum relative frequency for a character to be returned as the consensus character. If none, NA is returned.
`warn.non.IUPAC`	for the `IUPAC` method this argument is passed to `bma` with a default value set to FALSE to avoid warnings due to gap characters in the alignment.
`type`	for the `IUPAC` method this argument is passed to `bma`.

Details

"majority" The character with the higher frequency is returned as the consensus character.

"threshold" As above but in addition the character relative frequency must be higher than the value controled by the `threshold` argument. If none, NA id returned.

"IUPAC" Make sense only for nucleic acid sequences (DNA or RNA). The consensus character is defined if possible by an IUPAC symbol by function `bma`. If this is not possible, when there is a gap character for instance, NA is returned.

"profile" With this method a matrix with the count of each possible character at each position is returned.

`con` is a short form for `consensus`.

Value

Either a vector of single characters with possible NA or a matrix with the method `profile`.

Author(s)

J.R. Lobry

References

```
citation("seqinr")
```

See Also

See `read.alignment` to import alignment from files.

Examples

```
#
# Read 5 aligned DNA sequences at 42 sites:
#
  phylip <- read.alignment(file = system.file("sequences/test.phylip",
    package = "seqinr"), format = "phylip")
#
# Show data in a matrix form:
#
  (matali <- as.matrix(phylip))
#
# With the majority rule:
#
  res <- consensus(phylip)
  stopifnot(c2s(res) == "aaaccctggccgttcagggtaaaccgtggccgggcagggtat")
#
# With a threshold:
#
  res.thr <- consensus(phylip, method = "threshold")
  res.thr[is.na(res.thr)] <- "." # change NA into dots
```

```
  stopifnot(c2s(res.thr) == "aa.c..t.gc.gtt..g..t.a.cc..ggccg.......ta.")
#
# With an IUPAC summary:
#
  res.iup <- consensus(phylip, method = "IUPAC")
  stopifnot(c2s(res.iup) == "amvsbnkkgcmkkkmmgsktrmrssndkgcmrkdmmvskyaw")
  # replace 3 and 4-fold symbols by dots:
  res.iup[match(res.iup, s2c("bdhvn"), nomatch = 0) > 0] <- "."
  stopifnot(c2s(res.iup) == "am.s..kkgcmkkkmmgsktrmrss..kgcmrk.mm.skyaw")
#
# With a profile method:
#
  (res <- consensus(phylip, method = "profile"))
#
# Show the connection between the profile and some consensus:
#
  bxc <- barplot(res, col = c("green", "blue", "orange", "white", "red"), border = NA,
  space = 0, las = 2, ylab = "Base count",
  main = "Profile of a DNA sequence alignment",
  xlab = "sequence position", xaxs = "i")

  text(x = bxc, y = par("usr")[4],lab = res.thr, pos = 3, xpd = NA)
  text(x = bxc, y = par("usr")[1],lab = res.iup, pos = 1, xpd = NA)
```

`count` *Composition of dimer/trimer/etc oligomers*

Description

Counts the number of times dimer/trimer/etc oligomers occur in a sequence. Note that the oligomers are overlapping by default.

Usage

```
count(seq, word, start = 0, by = 1, freq = FALSE, alphabet = s2c("acgt"), frame = s
```

Arguments

`seq`	a vector of single characters.
`word`	an integer giving the size of word (n-mer) to count.
`start`	an integer (0, 1, 2,...) giving the starting position to consider in the sequence. The default value 0 means that we start at the first nucleotide in the sequence.
`by`	an integer defaulting to 1 for the window step.
`freq`	if TRUE, word relative frequencies (summing to 1) are returned instead of counts
`alphabet`	a vector of single characters used to build the oligomer set.
`frame`	synonymous for start

Details

`count` counts the occurence of all words by moving a window of length `word`. The window step is controlled by the argument `by`. `start` controls the starting position in the sequence for the count.

Value

This function returns a `table` whose `dimnames` are all the possible oligomers. All oligomers are returned, even if absent from the sequence.

Author(s)

D. Charif, J.R. Lobry with suggestions from Gabriel Valiente, Stefanie Hartmann and Christian Gautier

References

```
citation("seqinr")
```

See Also

`table` for the class of the returned objet. See `rho` and `zscore` for dinucleotide statistics.

Examples

```
a <- s2c("acgggtacggtcccatcgaa")
##
## To count dinucleotide occurrences in sequence a:
##
count(a, word = 2)
##
## To count trinucleotide occurrences in sequence a, with start = 2:
##
count(a, word = 3, start = 2)
##
## To count dinucleotide relative frequencies in sequence a:
##
count(a, word = 2, freq = TRUE)
##
## To count dinucleotides in codon positions III-I in a coding sequence:
##
alldinuclIIIpI <- s2c("NNaaNatNttNtgNgtNtcNctNtaNagNggNgcNcgNgaNacNccNcaNN")
resIIIpI <- count(alldinuclIIIpI, word = 2, start = 2, by = 3)
stopifnot(all( resIIIpI == 1))
##
## Simple sanity check:
##
alldinucl <- "aattgtctaggcgacca"
stopifnot(all(count(s2c(alldinucl), 2) == 1))
alldiaa <- "aaxxzxbxvxyxwxtxsxpxfxmxkxlxixhxgxexqxcxdxnxrxazzbzvzyzwztzszpzfzmzkzlzizhzgzezq
stopifnot(all(count(s2c(alldiaa), 2, alphabet = s2c("arndcqeghilkmfpstwyvbzx")) == 1))
##
```

```
## Example with dinucleotide count in the complete Human mitochondrion genome:
##
humanMito <- read.fasta(file = system.file("sequences/humanMito.fasta", package = "seqinr"))
##
## Get the dinucleotide count:
##
dinu <- count(humanMito[[1]], 2)
##
## Put the results in a 4 X 4 array:
##
dinu2 <- dinu
dim(dinu2) <- c(4, 4)
nucl <- s2c("ACGT")
dimnames(dinu2) <- list(paste(nucl, "-3\'", sep = ""), paste("5\'-", nucl, sep = ""))
##
## Show that CpG and GpT dinucleotides are depleted:
##
mosaicplot(t(dinu2), shade = TRUE,
  main = "Dinucleotide XpY frequencies in the Human\nmitochondrion complete genome",
  xlab = "First nucleotide: Xp",
  ylab = "Second nucleotide: pY", las = 1, cex = 1)
mtext("Note the depletion in CpG and GpT dinucleotides", side = 1, line = 3)
```

`countfreelists` *The number of free lists available and annotation lines in an ACNUC server*

Description

Returns the number of free lists available list of names of annotation lines in the opened ACNUC database.

Usage

```
countfreelists(socket = autosocket())
cfl(socket = autosocket())
```

Arguments

`socket`	an object of class `sockconn` connecting to a remote ACNUC database (default is a socket to the last opened database).

Value

a list with the following 2 components:

`free`	numeric. The number of free lists
`annotlines`	vector of strings. Names of annotation lines

Author(s)

J.R. Lobry

References

http://pbil.univ-lyon1.fr/databases/acnuc.html

```
citation("seqinr")
```

See Also

choosebank, query

Examples

```
## Not run:
 # Need internet connection
  choosebank("emblTP")
  (rescountfreelists <- countfreelists())
  stopifnot(all(rescountfreelists$annotlines ==
   c("ALL", "AC",  "PR",  "DT",  "KW",  "OS",  "OC",
   "OG",  "RN",  "RC",  "RP",  "RX", "RG",  "RA",  "RT",  "RL",  "DR",
   "CC",  "AH",  "AS",  "FH",  "FT",  "CO",  "SQ", "SEQ")))
  closebank()

## End(Not run)
```

---

countsubseqs *Number of subsequences in an ACNUC list*

Description

Returns the number of subsequences in the ACNUC list of rank `lrank`.

Usage

```
countsubseqs(lrank, socket = autosocket())
css(lrank, socket = autosocket())
```

Arguments

`lrank`	the rank of the ACNUC list to consider.
`socket`	an object of class `sockconn` connecting to a remote ACNUC database (default is a socket to the last opened database).

Value

Numeric.

Author(s)

J.R. Lobry

References

http://pbil.univ-lyon1.fr/databases/acnuc.html

```
citation("seqinr")
```

See Also

choosebank, query, glr to get a list rank from its name.

Examples

```
## Not run:
 # Need internet connection
  choosebank("emblTP")
  query("mylist", "N=@", virtual = TRUE) # select all (seqs + subseqs)
  mylist$nelem   # 14138094 seqs + subseqs
  stopifnot(mylist$nelem == 14138094)
  css(glr("mylist")) # 1604500 subsequences only
  stopifnot(css(glr("mylist")) == 1604500)
  closebank()

## End(Not run)
```

---

crelistfromclientdata

To create on server an ACNUC list from data lines sent by client

---

Description

This function is usefull if you have a local file with sequence names (sequence ID), or sequence accession numbers, or species names, or keywords. This allows you to create on the server a list with the corresponding items.

Usage

```
crelistfromclientdata(listname, file, type, socket = autosocket(), invisible = TRUE
clfcd(listname, file, type, socket = autosocket(), invisible = TRUE, verbose = FALS
```

Arguments

listname	The name of the list as a quoted string of chars
file	The local file name
type	Could be one of "SQ", "AC", "SP", "KW", see examples

socket	an object of class sockconn connecting to a remote ACNUC database (default is a socket to the last opened database).
invisible	if FALSE, the result is returned visibly.
verbose	if TRUE, verbose mode is on
virtual	if TRUE, no attempt is made to retrieve the information about all the elements of the list. In this case, the req component of the list is set to NA.

Details

clfcd is a shortcut for crelistfromclientdata.

Value

The result is directly assigned to the object listname in the user workspace. This is an objet of class qaw, a list with the following 6 components:

call	the original call
name	the ACNUC list name
nelem	the number of elements (for instance sequences) in the ACNUC list
typelist	the type of the elements of the list. Could be SQ for a list of sequence names, KW for a list of keywords, SP for a list of species names.
req	a list of sequence names that fit the required criteria or NA when called with parameter virtual is TRUE
socket	the socket connection that was used

Author(s)

J.R. Lobry

References

```
citation("seqinr")
```

See Also

choosebank, query, savelist for the reverse operation with an ACNUC list of sequences.

Examples

```
## Not run:
# Need internet connection
choosebank("emblTP")
#
# Example with a file that contains sequence names:
#
fileSQ <- system.file("sequences/bb.mne", package = "seqinr")
crelistfromclientdata("listSQ", file = fileSQ, type = "SQ")
sapply(listSQ$req, getName)
```

```
#
# Example with a file that contains sequence accession numbers:
#
fileAC <- system.file("sequences/bb.acc", package = "seqinr")
crelistfromclientdata("listAC", file = fileAC, type = "AC")
sapply(listAC$req, getName)
#
# Example with a file that contains species names:
#
fileSP <- system.file("sequences/bb.sp", package = "seqinr")
crelistfromclientdata("listSP", file = fileSP, type = "SP")
sapply(listSP$req, getName)
#
# Example with a file that contains keywords:
#
fileKW <- system.file("sequences/bb.kwd", package = "seqinr")
crelistfromclientdata("listKW", file = fileKW, type = "KW")
sapply(listKW$req, getName)
#
# Summary of ACNUC lists:
#
sapply(alr()$rank, getliststate)
closebank()

## End(Not run)
```

`dia.bactgensize` *Distribution of bacterial genome size from GOLD*

Description

This function tries to download the last update of the GOLD (Genomes OnLine Database) to extract bacterial genomes sizes when available. The histogram and the default density() output is produced. Optionally, a maximum likelihood estimate of a superposition of two or three normal distributions is also represented.

Usage

```
dia.bactgensize(fit = 2, p = 0.5, m1 = 2000, sd1 = 600, m2 = 4500,
       sd2 = 1000, p3 = 0.05, m3 = 9000, sd3 = 1000, maxgensize = 20000,
       source = c(system.file("sequences/goldtable15Dec07.txt", package = "seqinr")
    "http://www.genomesonline.org/DBs/goldtable.txt"))
```

Arguments

`fit`	integer value. If `fit == 0` no normal fit is produced, if `fit == 2` try to fit a superposition of two normal distributions, if `fit == 3` try to fit a superposition of three normal distributions.
`p`	initial guess for the proportion of the first population.

m1	initial guess for the mean of the first population.
sd1	initial guess for the standard deviation of the first population.
m2	initial guess for the mean of the second population.
sd2	initial guess for the standard deviation of the second population.
p3	initial guess for the proportion of the third population.
m3	initial guess for the mean of the third population.
sd3	initial guess for the standard deviation of the third population.
maxgensize	maximum admissive value in bp for a bacterial genome size: only value less or equal to this threshold are considrered.
source	the file with raw data. By default a local (outdated) copy is used.

Value

An invisible dataframe with three components:

genus	genus name
species	species names
gs	genome size in Kb

Author(s)

J.R. Lobry

References

Please cite the following references when using data from GOLD:

Kyrpides, N.C. (1999) Genomes OnLine Database (GOLD 1.0): a monitor of complete and ongoing genome projects world-wide. *Bioinformatics*, **15**:773-774.

Bernal, A., Ear, U., Kyrpides, N. (2001) Genomes OnLine Database (GOLD): a monitor of genome projects world-wide. *Nucleic Acids Research*, **29**:126-127.

Liolios, K., Tavernarakis, N., Hugenholtz, P., Kyrpides, N.C. (2006) The Genomes On Line Database (GOLD) v.2: a monitor of genome projects worldwide. *Nucleic Acids Research*, **34**:D332-D334.

Liolios, K., Mavrommatis, K., Tavernarakis, N., Kyrpides, N.C. (2008) The Genomes On Line Database (GOLD) in 2007: status of genomic and metagenomic projects and their associated metadata. *Nucleic Acids Research*, **in press**:D000-D000.

```
citation("seqinr")
```

See Also

density

Examples

```
#
# With a local outdated copy from GOLD:
#
   dia.bactgensize()
#
# With last GOLD data:
#
  ## Not run:
  # Need internet connection
  dia.bactgensize(source = "http://www.genomesonline.org/DBs/goldtable.txt")

## End(Not run)
```

`dinucl` *Mean zscore on 242 complete bacterial chromosomes*

Description

This dataset contains the mean zscores as computed on all intergenic sequences (intergenic) and on all CDS (coding) from 242 complete bacterial chromosomes (as retrieved from Genome Reviews database on June 16, 2005).

Usage

```
data(dinucl)
```

Format

List of two dataframes of 242 chromosomes and 16 dinucleotides: one for intergenic, one for coding sequences.

intergenic the mean of `zscore` computed with the `base` model on each intergenic sequence

coding the mean of `zscore` computed with the `codon` model on each coding sequence

References

Palmeira, L., Guéguen, L. and Lobry JR. (2006) UV-targeted dinucleotides are not depleted in light-exposed Prokaryotic genomes. *Molecular Biology and Evolution*, **23**:2214-2219.
http://mbe.oxfordjournals.org/cgi/reprint/23/11/2214

```
citation("seqinr")
```

See Also

`zscore`

Examples

```
data(dinucl)
par(mfrow = c(2, 2), mar = c(4,4,0.5,0.5)+0.1)
myplot <- function(x){
  plot(dinucl$intergenic[, x], dinucl$coding[, x],
  xlab = "intergenic", ylab = "coding",
  las = 1, ylim = c(-6, 4),
  xlim = c(-3, 3), cex = 0)
  rect(-10,-10,-1.96,10,col="yellow", border = "yellow")
  rect(1.96,-10,10,10,col="yellow", border = "yellow")
  rect(-10,-10,10,-1.96,col="yellow", border = "yellow")
  rect(-10,1.96,10,10,col="yellow", border = "yellow")
  abline(v=0,lty=3)
  abline(h=0,lty=3)
  abline(h=-1.96,lty=2)
  abline(h=+1.96,lty=2)
  abline(v=-1.96,lty=2)
  abline(v=+1.96,lty=2)
  points(dinucl$intergenic[, x], dinucl$coding[, x], pch = 21,
  col = rgb(.1,.1,.1,.5), bg = rgb(.5,.5,.5,.5))
  legend("bottomright", inset = 0.02, legend = paste(substr(x,1,1), "p", substr(x,2,2), " bi
  box()
}
myplot("CT")
myplot("TC")
myplot("CC")
myplot("TT")
```

---

dist.alignment *Pairwise Distances from Aligned Protein or DNA/RNA Sequences*

Description

These functions compute a matrix of pairwise distances from aligned sequences using similarity (Fitch matrix, for protein sequences only) or identity matrix (for protein and DNA sequences). The resulting matrix contains the squared root of the pairwise distances. For example, if identity between 2 sequences is 80 the squared root of (1.0 - 0.8) i.e. 0.4472136.

Usage

```
dist.alignment(x, matrix = c("similarity", "identity"))
```

Arguments

x	an object of class `alignment`, as returned by `read.alignment` for instance
matrix	the matrix distance to be used, partial matching allowed

Value

The distance matrix, object of class `dist`, computed by using the specified distance measure.

Author(s)

D. Charif and J.R. Lobry

References

The reference for the similarity matrix is :
Fitch, W.M. (1966) Mutation values for the interconversion of amino acid pair. *J. Mol. Biol.*, **16**:9-16.

```
citation("seqinr")
```

See Also

`read.alignment`

Examples

```
myseqs <- read.alignment(file = system.file("sequences/test.mase",
package = "seqinr"), format = "mase")
dist.alignment(myseqs, matrix = "identity" )
as.matrix(dist.alignment(myseqs, matrix = "identity" ))
```

---

dotPlot *Dot Plot Comparison of two sequences*

Description

Dot plots are most likely the oldest visual representation used to compare two sequences (see Maizel and Lenk 1981 and references therein). In its simplest form, a dot is produced at position (i,j) iff character number i in the first sequence is the same as character number j in the second sequence. More eleborated forms use sliding windows and a threshold value for two windows to be considered as matched.

Usage

```
dotPlot(seq1, seq2, wsize = 1, wstep = 1, nmatch = 1, col = c("white", "black"),
xlab = deparse(substitute(seq1)), ylab = deparse(substitute(seq2)), ...)
```

Arguments

seq1	the first sequence (x-axis) as a vector of single chars.
seq2	the second sequence (y-axis) as a vector of single char.
wsize	the size in chars of the moving window.
wstep	the size in chars for the steps of the moving window. Use `wstep == wsize` for non-overlapping windows.
nmatch	if the number of match per window is greater than or equal to `nmatch` then a dot is produced.
col	color of points passed to `image`.
xlab	label of x-axis passed to `image`.
ylab	label of y-axis passed to `image`.
...	further arguments passed to `image`.

Value

NULL.

Author(s)

J.R. Lobry

References

Maizel, J.V. and Lenk, R.P. (1981) Enhanced Graphic Matrix Analysis of Nucleic Acid and Protein Sequences. *Proceedings of the National Academy of Science USA*, **78**:7665-7669.

```
citation("seqinr")
```

See Also

image

Examples

```
#
# Identity is on the main diagonal:
#
dotPlot(letters, letters, main = "Direct repeat")
#
# Internal repeats are off the main diagonal:
#
dotPlot(rep(letters, 2), rep(letters, 2), main = "Internal repeats")
#
# Inversions are orthogonal to the main diagonal:
#
dotPlot(letters, rev(letters), main = "Inversion")
#
```

```
# Insertion in the second sequence yields a vertical jump:
#
dotPlot(letters, c(letters[1:10], s2c("insertion"), letters[11:26]),
  main = "Insertion in the second sequence", asp = 1)
#
# Insertion in the first sequence yields an horizontal jump:
#
dotPlot(c(letters[1:10], s2c("insertion"), letters[11:26]), letters,
  main = "Insertion in the first sequence", asp = 1)
#
# Protein sequences have usually a good signal/noise ratio because there
# are 20 possible amino-acids:
#
aafile <- system.file("sequences/seqAA.fasta", package = "seqinr")
protein <- read.fasta(aafile)[[1]]
dotPlot(protein, protein, main = "Dot plot of a protein\nwsize = 1, wstep = 1, nmatch = 1")
#
# Nucleic acid sequences have usually a poor signal/noise ratio because
# there are only 4 different bases:
#
dnafile <- system.file("sequences/malM.fasta", package = "seqinr")
dna <- protein <- read.fasta(dnafile)[[1]]
dotPlot(dna[1:200], dna[1:200], main = "Dot plot of a nucleic acid sequence\nwsize = 1, wste
#
# Play with the wsize, wstep and nmatch arguments to increase the
# signal/noise ratio:
#
dotPlot(dna[1:200], dna[1:200], wsize = 3, wstep = 3, nmatch = 3,
main = "Dot plot of a nucleic acid sequence\nwsize = 3, wstep = 3, nmatch = 3")
```

`dotchart.uco` *Cleveland plot for codon usage tables*

Description

Draw a Cleveland dot plot for codon usage tables

Usage

```
dotchart.uco(x, numcode = 1, aa3 = TRUE, cex = 0.7, alphabet = s2c("tcag"),
 pch = 21, gpch = 20, bg = par("bg"), color = par("fg"), gcolor = par("fg"),
lcolor = "gray", xlim, ...)
```

Arguments

`x`	table of codon usage as computed by `uco`.
`numcode`	the number of the code to be used by `translate`.
`aa3`	logical. If TRUE use the three-letter code for amino- acids. If FALSE use the one-letter code for amino-acids.

cex	the character size to be used.
alphabet	character for codons labels
pch	the plotting character or symbol to be used.
gpch	the plotting character or symbol to be used for group values.
bg	the background color to be used.
color	the color(s) to be used for points an labels.
gcolor	the single color to be used for group labels and values.
lcolor	the color(s) to be used for the horizontal lines.
xlim	horizontal range for the plot
...	graphical parameters can also be specified as arguments

Value

An invisible list with components:

x	table of codon usage
labels	codon names
groups	amino acid factor
gdata	sums by amino acid
ypg	the y-axis coordinates for amino acids
ypi	the y-axis coordinates for codons

Author(s)

J.R. Lobry

References

Cleveland, W. S. (1985) The Elements of Graphing Data. Monterey, CA: Wadsworth. `citation("seqinr")`

See Also

`dotchart`, `uco`, `aaa`, `translate`

Examples

```
# Load dataset:
data(ec999)
# Compute codon usage for all coding sequences:
ec999.uco <- lapply(ec999, uco, index="eff")
# Put it in a dataframe:
df <- as.data.frame(lapply(ec999.uco, as.vector))
# Add codon names:
row.names(df) <- names(ec999.uco[[1]])
# Compute global codon usage:
global <- rowSums(df)
# Choose a title for the graph:
```

```
title <- "Codon usage in 999 E. coli coding sequences"
# Plot data:
dotchart.uco(global, main = title)
```

`draw.oriloc` *Graphical representation for nucleotide skews in prokaryotic chromosomes.*

Description

Graphical representation for nucleotide skews in prokaryotic chromosomes.

Usage

```
draw.oriloc(ori, main = "Title",
  xlab = "Map position in Kb",
  ylab = "Cumulated combined skew in Kb", las = 1, las.right = 3,
  ta.mtext = "Cumul. T-A skew", ta.col = "pink", ta.lwd = 1,
  cg.mtext = "Cumul. C-G skew", cg.col = "lightblue", cg.lwd = 1,
  cds.mtext = "Cumul. CDS skew", cds.col = "lightgreen", cds.lwd = 1,
  sk.col = "black", sk.lwd = 2,
  add.grid = TRUE, ...)
```

Arguments

`ori`	A data frame obtained with the `oriloc` function.
`main`	The main title of the plot.
`xlab`	The x-axis title.
`ylab`	The y-axis title.
`las`	The style of axis labels for the bottom and left axes.
`las.right`	The style of axis labels for the right axis.
`ta.mtext`	The marginal legend for the TA skew.
`ta.col`	The color for the TA skew.
`ta.lwd`	The line width for the TA skew.
`cg.mtext`	The marginal legend for the CG skew.
`cg.col`	The color for the CG skew.
`cg.lwd`	The line width for the CG skew.
`cds.mtext`	The marginal legend for the CDS skew.
`cds.col`	The color for the CDS skew.
`cds.lwd`	The line width for the CDS skew.
`sk.col`	The color for the cumulated combined skew.
`sk.lwd`	The line width for the cumulated combined skew.
`add.grid`	Logical, if `TRUE` a vertical grid is added to the plot.
`...`	Further arguments are passed to the function `plot`.

Author(s)

Jean R. Lobry

References

```
citation("seqinr")
```

See Also

oriloc, rearranged.oriloc, extract.breakpoints

Examples

```
#
# Example with Chlamydia trachomatis complete genome
#
  ori <- oriloc()
  draw.oriloc(ori)
#
# The same, using more options from function draw.oriloc()
#
draw.oriloc(ori,
  main = expression(italic(Chlamydia~~trachomatis)~~complete~~genome),
  ta.mtext = "TA skew", ta.col = "red",
  cg.mtext = "CG skew", cg.col = "blue",
  cds.mtext = "CDS skew", cds.col = "seagreen",
  add.grid = FALSE)
```

---

draw.rearranged.oriloc

Graphical representation for rearranged nucleotide skews in prokaryotic chromosomes.

Description

Graphical representation for rearranged nucleotide skews in prokaryotic chromosomes.

Usage

```
draw.rearranged.oriloc(rearr.ori, breaks.gcfw = NA, breaks.gcrev = NA, breaks.atfw
```

Arguments

rearr.ori	A data frame obtained with the rearranged.oriloc function.
breaks.gcfw	The coordinates of the breakpoints in the GC-skew, for forward transcribed protein coding sequences. These coordinates can be obtained with the extract.breakpoints function.

`breaks.gcrev` The coordinates of the breakpoints in the GC-skew, for reverse transcribed protein coding sequences. These coordinates can be obtained with the `extract.breakpoints` function.

`breaks.atfw` The coordinates of the breakpoints in the AT-skew, for forward transcribed protein coding sequences. These coordinates can be obtained with the `extract.breakpoints` function.

`breaks.atrev` The coordinates of the breakpoints in the AT-skew, for reverse transcribed protein coding sequences. These coordinates can be obtained with the `extract.breakpoints` function.

Author(s)

Jean R. Lobry and A. Necsulea

References

Necsulea, A. and Lobry, J.R. (2007) A New Method for Assessing the Effect of Replication on DNA Base Composition Asymmetry. *Molecular Biology and Evolution*, **24**:2169-2179.

See Also

`rearranged.oriloc`, `extract.breakpoints`

Examples

```
## Not run:
### Example for Chlamydia trachomatis ####

### Rearrange the chromosome and compute the nucleotide skews ###

r.ori <- rearranged.oriloc(seq.fasta = system.file("sequences/ct.fasta", package = "seqinr")
    g2.coord = system.file("sequences/ct.coord", package = "seqinr"))

### Extract the breakpoints for the rearranged nucleotide skews ###

breaks <- extract.breakpoints(r.ori, type = c("gcfw", "gcrev"), nbreaks = c(2, 2), gridsize

### Draw the rearranged nucleotide skews and place the position of the breakpoints on the gr

draw.rearranged.oriloc(r.ori, breaks.gcfw = breaks$gcfw$breaks, breaks.gcrev = breaks$gcrev$
```

---

ec999 *999 coding sequences from E. coli*

Description

This dataset contains 999 coding sequences from the Escherichia coli chromosome

Usage

```
data(ec999)
```

Format

List of 999 vectors of characters, one for each coding sequence.

ECFOLE.FOLE chr [1:672] "A" "T" "G" "C" ...

ECMSBAG.MSBA chr [1:1749] "A" "T" "G" "C" ...

ECNARZYW-C.NARV chr [1:681] "A" "T" "G" "A" ...

... ... TRUNCATED ...

XYLEECOM.MALK chr [1:1116] "A" "T" "G" "G" ...

XYLEECOM.LAMB chr [1:1341] "A" "T" "G" "A" ...

XYLEECOM.MALM chr [1:921] "A" "T" "G" "A" ...

References

Lobry, J.R., Gautier, C. (1994) Hydrophobicity, expressivity and aromaticity are the major trends of amino-acid usage in 999 *Escherichia coli* chromosome-encode genes. *Nucleic Acids Research*,**22**:3174-3180.

```
citation("seqinr")
```

Examples

```
data(ec999)
```

---

extract.breakpoints

Extraction of breakpoint positions on the rearranged nucleotide skews.

Description

Extraction of breakpoint positions on the rearranged nucleotide skews.

Usage

```
extract.breakpoints(rearr.ori,type = c("atfw", "atrev", "gcfw", "gcrev"), nbreaks,
```

Arguments

rearr.ori	A data frame obtained with the rearranged.oriloc function.
type	The type of skew for which to extract the breakpoints; must be a subset of c("atfw","atrev","gcfw","gcrev").
nbreaks	The number of breakpoints to extract for each type of skew. Provide a vector of the same length as type.
gridsize	To make sure that the best breakpoints are found, and to avoid finding only a local extremum of the likelihood and residual sum of square functions, a grid search is performed. The search for breakpoints is repeated gridsize times, with different starting values for the breakpoints.
it.max	The maximum number of iterations to be performed when searching for the breakpoints. This argument corresponds to the it.max argument in segmented.

Details

This method uses the segmented function in the segmented package to extract the breakpoints positions in the rearranged nucleotide skews obtained with the rearranged.oriloc function. To make sure that the best breakpoints are found, and to avoid finding only a local extremum of the likelihood and residual sum of square functions, a grid search is performed. The search for breakpoints is repeated gridsize times, with different starting values for the breakpoints.

Value

This function returns a list, with as many elements as the type argument (for example $gcfw will contain the results for the rearranged GC-skew, for forward-encoded genes). Each element of this list is also a list, containing the following information: in $breaks the position of the breakpoints on the rearranged chromosome; in $slopes.left the slopes of the segments on the left side of each breakpoint; in $slopes.right the slopes of the segments on the right side of each breakpoint; in $real.coord, the coordinates of the breakpoints on the real chromosome (before rearrangement).

Author(s)

A. Necsulea

References

```
citation("segmented")
```

Necsulea, A. and Lobry, J.R. (in prep) A novel method for assessing the effect of replication on DNA base composition asymmetry.

See Also

oriloc, draw.rearranged.oriloc, rearranged.oriloc

Examples

```
### Example for Chlamydia trachomatis ####

### Rearrange the chromosome and compute the nucleotide skews ###

## Not run:
r.ori <- rearranged.oriloc(seq.fasta = system.file("sequences/ct.fasta",package = "seqinr"),
    g2.coord = system.file("sequences/ct.coord",package = "seqinr"))
## End(Not run)

### Extract the breakpoints for the rearranged nucleotide skews ###

## Not run: breaks <- extract.breakpoints(r.ori,type = c("gcfw", "gcrev"), nbreaks = c(2, 2)

### Draw the rearranged nucleotide skews and place the position of the breakpoints on the gr

## Not run: draw.rearranged.oriloc(r.ori, breaks.gcfw = breaks$gcfw$breaks, breaks.gcrev = b
```

`extractseqs` *To extract the sequences information of a sequence or a list of sequence in different formats*

Description

The function allows to extract large amount of data as whole genome sequences,using different output formats and types of extraction. This function is not yet available for windows.

Usage

```
extractseqs(listname,socket = autosocket(), format="fasta",operation="simple",featu
exseq(listname,socket = autosocket(), format="fasta",operation="simple", feature="x
```

Arguments

`listname`	the name of list on server (may be a virtual list)
`socket`	an object of class `sockconn` connecting to a remote ACNUC database (default is a socket to the last opened database).
`format`	the format of output.Can be `acnuc`, `fasta`,`flat` or `coordinates`
`operation`	the type of extraction. Can be `simple`, `translate`, `fragment`, `feature` or `region`
`feature`	-optional- the feature to be extracted (for operations "feature" or "region"): a feature table item (CDS, mRNA,...)
`bounds`	-optional- the bounds for extraction (for operations "fragment" or "region")
`minbounds`	-optional- the minimal bounds for extraction (for operations "fragment" or "region")

verbose	if TRUE, verbose mode is on
nzlines	number of line in zlib mode

Details

To extract a list of sequences (lrank argument) or a single sequence (seqnum argument) using different output formats and types of extraction. All formats except "coordinates" extract sequence data. Format "coordinates" extract coordinate data; start > end indicates the complementary strand.

listname sequence list name.

socket a socket of class connection and sockconn returned by choosebank. Default value (auto) means that the socket will be set to to the socket component of the banknameSocket variable.

format acnuc, fasta, flat or coordinates

operation simple, translate, fragment, feature or region

feature (for operations "feature" or "region") a feature table item (CDS, mRNA,...).

simple each sequence or subsequence is extracted.

translate meaningful only for protein-coding (sub)sequences that are extracted as protein sequences. Nothing is extracted for non-protein coding sequences.

fragment Allows to extract any part of the sequence(s) in list. Such part is specified by the bounds and minbounds arguments according to the syntax suggested by these examples:

132,1600	to extract from nucl. 132 to nucl 1600 of the sequence. If applied to a subsequence, coordinates are in the pare
-10,10	to extract from 10 nucl. BEFORE the 5' end of the sequence to nucl. 10 of it. Useful only for subsequences, a
e-20,e+10	to extract from 20 nucl. BEFORE the 3' end of the sequence to 10 nucl. AFTER its 3' end. Useful only for sub
-20,e+5	to extract from 20 nucl. BEFORE the 5' end of the sequence to 5 nucl. AFTER its 3' end.

bounds (for operations "fragment" or "region") see syntax above.

minbounds same syntax as bounds. When the sequence data is too short for this quantity to be extracted, nothing is extracted. When the sequence data is between minbounds and bounds, extracted sequence data is extended by N's to the desired length.

Value

Sequence data.

Author(s)

S.Penel

References

```
citation("seqinr")
```

See Also

choosebank, query getlistrank

Examples

```
 ## Not run:
# Need internet connection
 choosebank("swissprotTP")
 query("mylist", "k=globin", virtual = TRUE)
 mylist.fasta <- exseq("mylist", verbose = TRUE)
 # 103 lines of FASTA
 stopifnot(length(mylist.fasta) == 103)
 closebank()

## End(Not run)
```

---

gb2fasta *conversion of GenBank file into fasta file*

Description

Converts a single entry in GenBank format into a fasta file.

Usage

```
gb2fasta(source.file =
"ftp://ftp.ncbi.nih.gov/genomes/Bacteria/Agrobacterium_tumefaciens_C58_Cereon/NC_00
destination.file = "Agrobacterium_tumefaciens_C58_Cereon.fasta")
```

Arguments

`source.file`	GenBank file
`destination.file`	Fasta file

Details

Multiple entries in GenBank file are not supported.

Value

none

Author(s)

J.R. Lobry

References

```
citation("seqinr")
```

See Also

oriloc

Examples

```
## Not run: gb2fasta()
```

---

gbk2g2 *Conversion of a GenBank format file into a glimmer-like one*

Description

This function reads a file in GenBank format and converts the features corresponding to CDS (Coding Sequences) into a format similar to glimmer program output.

Usage

```
gbk2g2(gbkfile = system.file("sequences/ct.gbk", package ="seqinr"),
g2.coord = "g2.coord")
```

Arguments

gbkfile	The name of the GenBank file
g2.coord	The name of the output file in glimmer-like format

Details

Partial CDS (either 5' or 3') and join in features are discarded.

Value

The input file is returned invisibly.

Author(s)

J.R. lobry

References

```
citation("seqinr")
```

See Also

oriloc which uses glimmer-like files, gbk2g2.euk for eukaryotic sequences with introns.

Examples

```
suppressWarnings(gbk2g2(g2.coord = "gbk2g2.test"))
res <- read.table("gbk2g2.test")
head(res)
stopifnot(nrow(res) == 892)
```

`gbk2g2.euk` *Conversion of a GenBank format file into a glimmer-like one. Eukaryotic version.*

Description

This function reads a file in GenBank format and converts the features corresponding to CDS (Coding Sequences) into a format similar to glimmer program output. This function is specifically made for eukaryotic sequences, i.e. with introns.

Usage

```
gbk2g2.euk(gbkfile = system.file("sequences/ame1.gbk", package ="seqinr"),
g2.coord = "g2.coord")
```

Arguments

`gbkfile`	The name of the GenBank file
`g2.coord`	The name of the output file

Details

This function returns the coordinates of the exons annotated in the GenBank format file.

Value

A data frame with three columns will be written to the `g2.coord` file. The first column corresponds to the name of the gene, given in the GenBank file through the `/gene` feature. The second and third column contain the start and the stop position of the exon.

Author(s)

J.R. Lobry and A. Necsulea

References

```
citation("seqinr")
```

See Also

`oriloc`, `gbk2g2`

Examples

```
## Not run:  gbk2g2.euk()
```

`get.db.growth` *Get the exponential growth of nucleic acid database content*

Description

Connects to the embl database to read the last release note about the number of nucleotides in the DDBJ/EMBL/Genbank database content. A log-linear fit is represented by dia.bd.gowth() with an estimate of the doubling time in months.

Usage

```
get.db.growth(where = "http://www.ebi.ac.uk/embl/Documentation/Release_notes/currer
dia.db.growth( get.db.growth.out = get.db.growth(), Moore = TRUE, ... )
```

Arguments

`where`	the file containig the database growth table.
`get.db.growth.out`	the output from get.db.growth()
`Moore`	logical, if TRUE add lines corresponding to an exponential growth rate with a doubling time of 18 months, that is Moore's law.
`...`	further arguments to plot

Value

A dataframe with the statistics from the embl site.

Author(s)

J.R. Lobry

References

http://www.ebi.ac.uk/embl/Documentation/Release_notes/current/relnotes.txt

```
citation("seqinr")
```

Examples

```
## Not run: data <- get.db.growth()
## Not run: dia.db.growth(data)
```

`get.ncbi` *Bacterial complete genome data from ncbi ftp site*

Description

Try to connect to ncbi ftp site to get a list of complete bacterial genomes.

Usage

```
get.ncbi(repository = "ftp://ftp.ncbi.nih.gov/genomes/Bacteria/")
```

Arguments

`repository`	Where to look for data. The default value is the location of the complete bacterial genome sequences at ncbi ftp repository.

Value

Returns a data frame which contains the following columns:

`species`	The species name as given by the corresponding folder name in the repository (*e.g.* Yersinia_pestis_KIM).
`accession`	The accession number as given by the common prefix of file names in the repository (*e.g.* NC_004088).
`size.bp`	The size of the sequence in bp (*e.g.* 4600755).
`type`	A factor with two levels (plasmid or chromosome) temptatively deduced from the description of the sequence.

WARNING

This function is highly dependant on ncbi ftp site conventions for which we have no control. The ftp connection apparently does not work when there is a proxy, this problem is circumvented here in a rather crude way.

Author(s)

J.R. Lobry

References

```
citation("seqinr")
```

Examples

```
## Not run: bacteria <- get.ncbi()
## Not run: summary(bacteria)
```

getAnnot *Generic Function to get sequence annotations*

Description

Annotations are taken from the Annot attribute for sequences imported from a FASTA file and retrieved from an ACNUC server for objects of the SeqAcnucWeb class.

Usage

```
getAnnot(object, ...)
## S3 method for class 'SeqAcnucWeb':
getAnnot(object, ..., nbl = 100, socket = autosocket())
```

Arguments

object	an object of the class SeqAcnucWeb or SeqFastadna, or SeqFastaAA or a list of these objects
nbl	the maximum number of line of annotation to read. Reading of lines stops when nbl lines have been transmitted or at the last annotation line of the sequence (SQ or ORIGIN line).
socket	an object of class sockconn connecting to a remote ACNUC database (default is a socket to the last opened database).
...	further arguments passed to or from other methods

Value

getAnnot returns a vector of string of characters containing the annotations for the sequences.

Author(s)

D. Charif and J.R. Lobry and L. Palmeira

References

```
citation("seqinr")
```

See Also

query, SeqAcnucWeb, c2s, translate and \code{translate} to select the annotation lines.

Examples

```
#
# List all available methods for getAnnot generic function:
#
   methods(getAnnot)
#
# SeqAcnucWeb class example:
#
  ## Not run:
  # Need internet connection
  choosebank("emblTP")
  query("fc", "sp=felis catus et t=cds et O=mitochondrion et Y>2001 et no k=partial")
  # get the first 5 lines annotating the first sequence:
  annots <- getAnnot(fc$req[[1]], nbl = 5)
  cat(annots, sep = "\n")
  # or use the list method to get them all at once:
  annots <- getAnnot(fc$req, nbl = 5)
  cat(annots, sep = "\n")
  closebank()

## End(Not run)
#
# SeqFastaAA class example:
#
   aafile <- system.file("sequences/seqAA.fasta", package = "seqinr")
   sfaa <- read.fasta(aafile, seqtype = "AA")
   getAnnot(sfaa[[1]])
#
# SeqFastadna class example:
#
   dnafile <- system.file("sequences/malM.fasta", package = "seqinr")
   sfdna <- read.fasta(file = dnafile)
   getAnnot(sfdna[[1]])
#
# Example with a FASTA file with multiple entries:
#
  ff <- system.file("sequences/someORF.fsa", package = "seqinr")
  fs <- read.fasta(ff)
  getAnnot(fs) # the list method is used here to get them all at once
#
# Default getAnnot method example. An error is produced because
# there are no annotations by default:
#
   result <- try(getAnnot(letters))
   stopifnot(!inherits("result", "try-error"))
```

`getFrag` *Generic function to extract sequence fragments*

Description

getFrag is used to extract the sequence fragment starting at the `begin` position and ending at the `end` position.

Usage

```
getFrag(object, begin, end, ...)
## S3 method for class 'SeqAcnucWeb':
getFrag(object, begin, end, ..., socket = autosocket(), name = getName(object))
## S3 method for class 'SeqFastadna':
getFrag(object, begin, end, ..., name = getName(object))
## S3 method for class 'SeqFastaAA':
getFrag(object, begin, end, ..., name = getName(object))
## S3 method for class 'SeqFrag':
getFrag(object, begin, end, ..., name = getName(object))
```

Arguments

`object`	an object of the class `SeqAcnucWeb` or `SeqFastadna`, or `SeqFastaAA` or `SeqFrag` or a list of these objects
`begin`	First position of the fragment to extract. This position is included. Numerotation starts at 1.
`end`	Last position of the fragment to extract. This position is included.
`socket`	an object of class `sockconn` connecting to a remote ACNUC database (default is a socket to the last opened database by `choosebank`).
`name`	the sequence name
`...`	further arguments passed to or from other methods

Value

`getFrag` returns an object of class `SeqFrag`.

Author(s)

D. Charif and J.R. Lobry and L. Palmeira

References

```
citation("seqinr")
```

See Also

`SeqAcnucWeb`, `SeqFastadna`, `SeqFastaAA`, `SeqFrag`

Examples

```
#
# List all available methods for getFrag generic function:
#
   methods(getFrag)
#
# Example with a DNA sequence from a FASTA file:
#
  dnafile <- system.file("sequences/malM.fasta", package = "seqinr")
  sfdna <- read.fasta(file = dnafile)
  myfrag <- getFrag(sfdna[[1]], begin = 1, end = 10)
  stopifnot(getSequence(myfrag, as.string = TRUE) == "atgaaaatga")
```

---

getKeyword *Generic function to get keywords associated to sequences*

Description

Get keywords from an ACNUC server.

Usage

```
getKeyword(object, ...)
## S3 method for class 'SeqAcnucWeb':
getKeyword(object, ..., socket = autosocket())
```

Arguments

object	an object of the class SeqAcnucWeb, or a list of them, or the object resulting from query
socket	an object of class sockconn connecting to a remote ACNUC database (default is a socket to the last opened database by choosebank).
...	further arguments passed to or from other methods

Value

getKeyword returns a vector of strings containing the keyword(s) associated to a sequence.

Author(s)

D. Charif and J.R. Lobry and L. Palmeira

References

```
citation("seqinr")
```

See Also

SeqAcnucWeb

Examples

```
#
# List all available methods for getKeyword generic function:
#
   methods(getKeyword)
#
# Example of keyword extraction from an ACNUC server:
#
  ## Not run:
  # Need internet connection
  choosebank("emblTP")
  query("fc", "sp=felis catus et t=cds et o=mitochondrion")
  getKeyword(fc$req[[1]])
  # Should be:
  # [1] "DIVISION ORG" "RELEASE 62"   "CYTOCHROME B" "SOURCE"       "CDS"
  closebank()
## End(Not run)
```

---

getLength *Generic function to get the length of sequences*

Description

getLength returns the total number of bases or amino-acids in a sequence.

Usage

```
getLength(object, ...)
```

Arguments

object	an object of the class SeqAcnucWeb or SeqFastadna, or SeqFastaAA or SeqFrag or a list of these objects
...	further arguments passed to or from other methods

Value

getLength returns a numeric vector giving the length of the sequences.

Author(s)

D. Charif and J.R. Lobry and L. Palmeira

References

```
citation("seqinr")
```

See Also

SeqAcnucWeb, SeqFastadna, SeqFastaAA, SeqFrag

Examples

```
#
# List all available methods for getLength generic function:
#
   methods(getLength)
#
# Example with seven DNA sequences from a FASTA file:
#
  ff <- system.file("sequences/someORF.fsa", package = "seqinr")
  fs <- read.fasta(file = ff)
  stopifnot(all(getLength(fs) == c(5573, 5825, 2987, 3929, 2648, 2597, 2780)))
#
# Example with 49 sequences from an ACNUC server:
#
  ## Not run:
  # Need internet connection
  choosebank("emblTP")
  query("fc", "sp=felis catus et t=cds et o=mitochondrion")
  getLength(fc)
  closebank()
## End(Not run)
```

---

getLocation *Generic function to get the location of subsequences on the parent sequence*

Description

This function works only with subsequences from an ACNUC server.

Usage

```
getLocation(object, ...)
## S3 method for class 'SeqAcnucWeb':
getLocation(object, ..., socket = autosocket())
```

Arguments

object	an object of the class SeqAcnucWeb, or a list of them, or an object created by query
socket	an object of class sockconn connecting to a remote ACNUC database (default is a socket to the last opened database by choosebank).
...	further arguments passed to or from other methods

Value

A list giving the positions of the sequence on the parent sequence. If the sequence is a subsequence (*e.g.* coding sequence), the function returns the position of each exon on the parent sequence. NA is returned for parent sequences and a warning is isued.

Author(s)

D. Charif and J.R. Lobry and L. Palmeira

References

```
citation("seqinr")
```

See Also

SeqAcnucWeb

Examples

```
#
# List all available methods for getLocation generic function:
#
   methods(getLocation)
#
# Example with a subsequence from an ACNUC server:
#
  ## Not run:
  # Need internet connection
  choosebank("emblTP")
  query("fc", "sp=felis catus et t=cds et o=mitochondrion")
  getLocation(fc$req[[5]])
  closebank()
## End(Not run)
```

getName *Generic function to get the names of sequences*

Description

GetName returns the sequence names.

Usage

```
getName(object, ...)
```

Arguments

object	an object of the class SeqAcnucWeb or SeqFastadna, or SeqFastaAA or SeqFrag or a list of these objects
...	further arguments passed to or from other methods

Value

an object of class character containing the names of the sequences

Author(s)

D. Charif and J.R. Lobry and L. Palmeira

References

```
citation("seqinr")
```

See Also

SeqAcnucWeb, SeqFastadna, SeqFastaAA, SeqFrag

Examples

```
#
# List all available methods for getName generic function:
#
   methods(getName)
#
# Example with seven DNA sequences from a FASTA file:
#
  ff <- system.file("sequences/someORF.fsa", package = "seqinr")
  fs <- read.fasta(file = ff)
  stopifnot(all(getName(fs) == c("YAL001C", "YAL002W", "YAL003W",
    "YAL005C", "YAL007C", "YAL008W", "YAL009W")))
#
# Example with 49 sequences from an ACNUC server:
#
```

```
  ## Not run:
  # Need internet connection
  choosebank("emblTP")
  query("fc", "sp=felis catus et t=cds et o=mitochondrion")
  getName(fc)
  closebank()
## End(Not run)
```

`getSequence` *Generic function to get sequence data*

Description

getSequence returns the sequence either as vector of single chararacters or as a single string of multiple characters.

Usage

```
getSequence(object, as.string = FALSE, ...)
## S3 method for class 'SeqAcnucWeb':
getSequence(object, as.string = FALSE, ..., socket = autosocket())
```

Arguments

`object`	an object of the class `SeqAcnucWeb` or `SeqFastadna`, or `SeqFastaAA` or `SeqFrag` or a list of these objects, or an object of class `qaw` created by `query`
`as.string`	if TRUE sequences are returned as strings of multiple characters instead of a vector of single characters
`socket`	an object of class `sockconn` connecting to a remote ACNUC database (default is a socket to the last opened database).
`...`	further arguments passed to or from other methods

Value

For a single sequence an object of class `character` containing the characters of the sequence, either of length 1 when `as.string` is TRUE, or of the length of the sequence when `as.string` is FALSE. For many sequences, a list of these.

Author(s)

D. Charif and J.R. Lobry and L. Palmeira

References

```
citation("seqinr")
```

See Also

SeqAcnucWeb, SeqFastadna, SeqFastaAA, SeqFrag

Examples

```
#
# List all available methods for getSequence generic function:
#
   methods(getSequence)
#
# SeqAcnucWeb class example:
#
  ## Not run:
# Need internet connection
  choosebank("emblTP")
  query("fc", "sp=felis catus et t=cds et o=mitochondrion")
  getSequence(fc$req[[1]])
  getSequence(fc$req[[1]], as.string = TRUE)
  closebank()

## End(Not run)
#
# SeqFastaAA class example:
#
  aafile <- system.file("sequences/seqAA.fasta", package = "seqinr")
  sfaa <- read.fasta(aafile, seqtype = "AA")
  getSequence(sfaa[[1]])
  getSequence(sfaa[[1]], as.string = TRUE)
#
# SeqFastadna class example:
#
  dnafile <- system.file("sequences/someORF.fsa", package = "seqinr")
  sfdna <- read.fasta(file = dnafile)
  getSequence(sfdna[[1]])
  getSequence(sfdna[[1]], as.string = TRUE)
#
# SeqFrag class example:
#
  sfrag <- getFrag(object = sfdna[[1]], begin = 1, end = 10)
  getSequence(sfrag)
  getSequence(sfrag, as.string = TRUE)
```

---

getTrans *Generic function to translate coding sequences into proteins*

Description

This function translates nucleic acid sequences into the corresponding peptide sequence. It can translate in any of the 3 forward or three reverse sense frames. In the case of reverse sense, the

reverse-complement of the sequence is taken. It can translate using the standard (universal) genetic code and also with non-standard codes. Ambiguous bases can also be handled.

Usage

```
getTrans(object, sens = "F", NAstring = "X", ambiguous = FALSE, ...)
## S3 method for class 'SeqAcnucWeb':
getTrans(object, sens = "F", NAstring = "X", ambiguous = FALSE, ..., frame = "auto"
## S3 method for class 'SeqFastadna':
getTrans(object, sens = "F", NAstring = "X", ambiguous = FALSE, ..., frame = 0, num
## S3 method for class 'SeqFrag':
getTrans(object, sens = "F", NAstring = "X", ambiguous = FALSE, ..., frame = 0, num
```

Arguments

`object`	an object of the class `SeqAcnucWeb` or `SeqFastadna`, or `SeqFrag` or a list of these objects, or an object of class `qaw` created by `query`
`numcode`	The ncbi genetic code number for translation. By default the standard genetic code is used, and for sequences coming from an ACNUC server the relevant genetic code is used by default.
`NAstring`	How to translate amino-acids when there are ambiguous bases in codons.
`ambiguous`	If TRUE, ambiguous bases are taken into account so that for instance GGN is translated to Gly in the standard genetic code.
`frame`	Frame(s) (0,1,2) to translate. By default the frame `0` is used.
`sens`	Direction for translation: `F` for the direct strand e and `R` for the reverse complementary strand.
`...`	further arguments passed to or from other methods

Details

The following genetic codes are described here. The number preceding each code corresponds to `numcode`.

1 standard

2 vertebrate.mitochondrial

3 yeast.mitochondrial

4 protozoan.mitochondrial+mycoplasma

5 invertebrate.mitochondrial

6 ciliate+dasycladaceal

9 echinoderm+flatworm.mitochondrial

10 euplotid

11 bacterial+plantplastid

12 alternativeyeast

13 ascidian.mitochondrial

14 alternativeflatworm.mitochondrial

15 blepharism

16 chlorophycean.mitochondrial

21 trematode.mitochondrial

22 scenedesmus.mitochondrial

23 hraustochytrium.mitochondria

Value

For a single sequence an object of class `character` containing the characters of the sequence, either of length 1 when `as.string` is TRUE, or of the length of the sequence when `as.string` is FALSE. For many sequences, a list of these.

Author(s)

D. Charif and J.R. Lobry and L. Palmeira

References

```
citation("seqinr")
```

See Also

SeqAcnucWeb, SeqFastadna, SeqFrag\cr The genetic codes are given in the object \code{SeqFrag}, a more human readable form is given by the function \code{SeqFrag}. Use \code{SeqFrag} to get the three-letter code for amino-acids.

Examples

```
#
# List all available methods for getTrans generic function:
#
   methods(getTrans)
#
# Toy CDS example invented by Leonor Palmeira:
#
  toycds <- s2c("tctgagcaaataaatcgg")
  getTrans(toycds) # should be c("S", "E", "Q", "I", "N", "R")
#
# Toy CDS example with ambiguous bases:
#
  toycds2 <- s2c("tcngarcarathaaycgn")
  getTrans(toycds2) # should be c("X", "X", "X", "X", "X", "X")
```

```
  getTrans(toycds2, ambiguous = TRUE) # should be c("S", "E", "Q", "I", "N", "R")
  getTrans(toycds2, ambiguous = TRUE, numcode = 2) # should be c("S", "E", "Q", "X", "N", "R
#
# Real CDS example:
#
  realcds <- read.fasta(file = system.file("sequences/malM.fasta", package ="seqinr"))[[1]]
  getTrans(realcds)
# Biologically correct, only one stop codon at the end
  getTrans(realcds, frame = 3, sens = "R", numcode = 6)
# Biologically meaningless, note the in-frame stop codons

#
# Complex transsplicing operations, the correct frame and the correct
# genetic code are automatically used for translation into protein for
# sequences coming from an ACNUC server:
#
## Not run:
  # Need internet connection.
  # Translation of the following EMBL entry:
  #
  # FT   CDS             join(complement(153944..154157),complement(153727..153866),
  # FT                   complement(152185..153037),138523..138735,138795..138955)
  # FT                   /codon_start=1
  choosebank("emblTP")
  query("trans", "N=AE003734.PE35")
  getTrans(trans$req[[1]])
## End(Not run)
```

`getType` *To get available subsequence types in an opened ACNUC database*

Description

This function returns all subsequence types (e.g. CDS, TRNA) present in an opened ACNUC database, using default database if no socket is provided.

Usage

```
getType(socket = autosocket())
```

Arguments

`socket`	an object of class `sockconn` connecting to a remote ACNUC database (default is a socket to the last opened database).

Value

a list containing a short description for each subsequence type.

Author(s)

D. Charif and J.R. Lobry

References

```
citation("seqinr")
```

See Also

choosebank, query

Examples

```
## Not run:
# Need internet connection
  choosebank("emblTP")
  getType()
## End(Not run)
```

---

getlistrank *To get the rank of a list from its name*

Description

This is a low level function to get the rank of a list on server from its name.

Usage

```
getlistrank(listname, socket = autosocket(), verbose = FALSE)
glr(listname, socket = autosocket(), verbose = FALSE)
```

Arguments

listname	the name of list on server
socket	an object of class sockconn connecting to a remote ACNUC database (default is a socket to the last opened database).
verbose	if TRUE, verbose mode is on

Details

This low level function is usually not used directly by the user.

Value

The rank of list named listname on server, or 0 if no list with this name exists.

Author(s)

J.R. Lobry

References

```
citation("seqinr")
```

See Also

choosebank, query

Examples

```
## Not run:
# Need internet connection
choosebank("emblTP")
query("MyListName", "sp=Borrelia burgdorferi", virtual = TRUE)
(result <- getlistrank("MyListName"))
stopifnot(result == 2)
closebank()

## End(Not run)
```

---

getliststate *Asks for information about an ACNUC list of specified rank*

Description

Reply gives the type of list, its name, the number of elements it contains, and, for sequence lists, says whether the list contains only parent seqs (locus=T).

Usage

```
getliststate(lrank, socket = autosocket())
gls(lrank, socket = autosocket())
gln(lrank, ...)
```

Arguments

lrank	the name of the ACNUC list to modify
socket	an object of class sockconn connecting to a remote ACNUC database (default is a socket to the last opened database).
...	arguments passed to getliststate

Value

NA in case of problem and an warning is issued. When there is no problem a list with the following 4 components:

type	string. Type of ACNUC list (SQ, KW, SP)
name	string. ACNUC list name
count	numeric. Number of elements in ACNUC list
locus	logical. For ACNUC sequence lists TRUE means that the list contains only parent sequences. NA otherwise.

gln is a shortcut for getliststate(lrank, ...)$name

Author(s)

J.R. Lobry

References

http://pbil.univ-lyon1.fr/databases/acnuc.html

citation("seqinr")

See Also

choosebank, query, alr, glr

Examples

```
## Not run:

  ### Need internet connection
  choosebank("emblTP")
  query("mylist", "sp=felis catus et t=cds", virtual=TRUE)
  getliststate(glr("mylist")) # SQ, MYLIST, 603, FALSE
  gln(glr("mylist")) # MYLIST (upper case letters on server)
  closebank()

## End(Not run)
```

---

gfrag *Extract sequence identified by name or by number from an ACNUC server*

Description

Get length characters from sequence identified by name or by number starting from position start (counted from 1).

Usage

```
gfrag(what, start, length, idby = c("name", "number"), socket = autosocket())
```

Arguments

what	A sequence name or number
start	Start position from 1
length	Number of requested characters (answer may be shorter)
idby	Is the sequence identified by name or number? Default to name
socket	an object of class sockconn connecting to a remote ACNUC database (default is a socket to the last opened database).

Value

A string of characters with at most length characters (may be shorter than asked for). NA is returned and a warning is issued in case of problem (non existent sequence for instance).

Author(s)

J.R. Lobry

References

http://pbil.univ-lyon1.fr/databases/acnuc.html

```
citation("seqinr")
```

See Also

choosebank, query

Examples

```
## Not run:
# Need internet connection
  choosebank("emblTP")
  gfrag("LMFLCHR36", start = 1, length = 3529852) -> myseq
  stopifnot(nchar(myseq) == 3529852)
  closebank()

## End(Not run)
```

ghelp *Get help from an ACNUC server*

Description

Reads one item of information in specified help file from an ACNUC server. The are differences between ACNUC clients so that this help could be confusing. However, the query language is common to all clients so that the most recent documentation is most likely here.

Usage

```
ghelp(item = c("GENERAL", "SELECT", "SPECIES", "KEYWORD"), file = c("HELP", "HELP_W
```

Arguments

item	the name of the desired help item
file	the name of the help file on server side.
socket	an object of class sockconn connecting to a remote ACNUC database (default is a socket to the last opened database).
catresult	logical. If TRUE output is redirected to the console.

Value

A vector of string which is returned invisibly and "cated" to the console by default.

Author(s)

J.R. Lobry

References

http://pbil.univ-lyon1.fr/databases/acnuc.html

```
citation("seqinr")
```

See Also

choosebank, query

Examples

```
## Not run:

  ### Need internet connection
  choosebank("emblTP")
  ghelp()
  ghelp("SELECT")
```

```
  # To get info about current database:
  ghelp("CONT")

## End(Not run)
```

`isenum` *Get the ACNUC number of a sequence from its name or accession number*

Description

Gives the ACNUC number of a sequence in the `number` element of the returned list. More informations are returned for subsequences corresponding to coding sequences.

Usage

```
isenum(what, idby = c("name", "access"), socket = autosocket())
isn(what, ...)
getNumber.socket(socket, name)
getAttributsocket(socket, name)
```

Arguments

`what`	a sequence name or a sequence accession number
`idby`	is the sequence identified by name or by accession number? Default to name
`socket`	an object of class `sockconn` connecting to a remote ACNUC database (default is a socket to the last opened database).
`...`	arguments passed to `isenum`.
`name`	a sequence name.

Value

A list whith the following 6 components:

`number`	numeric. The ACNUC number of the sequence.
`length`	numeric. The length of the sequence.
`frame`	numeric. The reading frame (0, 1, or 2) of the sequence for CDS.
`gencode`	numeric. ACNUC's genetic code (0 means universal) of the sequence for CDS.
`ncbigc`	numeric. NCBI's genetic code (0 means universal) of the sequence for CDS.
`otheraccessmatches`	logical. If TRUE it means that several sequences are attached to the given accession nunmber, and that only the ACNUC number of the first attached sequence is returned in the `number` component of the list.

`isn(what, ...)` is a shortcut for `isenum(what, ...)$number`.

As from seqinR 1.1-3 `getNumber.socket` and `getAttributsocket` are deprecated (a warning is issued).

Author(s)

J.R. Lobry

References

http://pbil.univ-lyon1.fr/databases/acnuc.html

```
citation("seqinr")
```

See Also

choosebank, query

Examples

```
## Not run:

  ### Need internet connection
  choosebank("emblTP")
  isenum("LMFLCHR36")
  isn("LMFLCHR36")
  stopifnot(isn("LMFLCHR36") == 13682678)
  # Example with CDS:
  isenum("AB004237")

## End(Not run)
```

---

kaks *to Get an Estimation of Ka and Ks*

Description

Ks and Ka are respectively the number of substitutions per synonymous site and per nonsynonymous site between two protein-coding genes. The ratio of nonsynonymous (Ka) to synonymous (Ks) nucleotide substitution rates is an indicator of selective pressures on genes. A ratio significantly greater than 1 indicates positive selective pressure. A ratio around 1 indicates either neutral evolution at the protein level or an averaging of sites under positive and negative selective pressures. A ratio less than 1 indicates pressures to conserve protein sequence (i.e. purifying selection). This function estimates the Ka and Ks values for a set of aligned sequences using the method published by Li (1993) and gives the associated variance matrix.

Usage

```
kaks(x, debug = FALSE, forceUpperCase = TRUE)
```

Arguments

x	An object of class alignment
debug	If TRUE turns debug mode on
forceUpperCase	If TRUE, the default value, all character in sequences are forced to the upper case if at least one 'a', 'c', 'g', or 't' is found in the sequences. Turning it to FALSE if the sequences are already in upper case will save time.

Value

ks	matrix of Ks values
ka	matrix of Ka values
vks	variance matrix of Ks
vka	variance matrix of Ka

Note

When the alignment does not contain enough information (i.e we approach saturation), the Ka and Ks values take the value 10. Negative values indicate that Ka and Ks can not be computed.
Codons with ambiguous bases are treated as gaps.
Codons with gaps are not used for computations.

Author(s)

D. Charif, J.R. Lobry

References

Li, W.-H. (1993) Unbiased estimation of the rates of synonymous and nonsynonymous substitution. *J. Mol. Evol.*, **36**:96-99.
Hurst, L.D. (2002) The Ka/Ks ratio: diagnosing the form of sequence evolution. *Trends Genet.*, **18**:486-486.
The C programm implementing this method was provided by Manolo Gouy. More info is needed here to trace back the original C source so as to credit correct source. The original FORTRAN-77 code by Chung-I Wu modified by Ken Wolfe is available here http://wolfe.gen.tcd.ie/lab/pub/li93/.
For a recent discussion about the estimation of Ka and Ks see:
Tzeng, Y.H., Pan, R., Li, W.-H. (2004) Comparison of three methods for estimating rates of synonymous and nonsynonymous nucleotide substitutions. *Mol. Biol. Evol*, **21**:2290-2298.
The method implemented here is noted LWL85 in the above paper.
`citation("seqinr")`

See Also

read.alignment

Examples

```
#
# Simple Toy example:
#
s <- read.alignment(file = system.file("sequences/test.phylip", package = "seqinr"), format
kaks(s)
#
# Check numeric results on an simple test example:
#
data(AnoukResult)
Anouk <- read.alignment(file = system.file("sequences/Anouk.fasta", package = "seqinr"), fo
if( ! all.equal(kaks(Anouk), AnoukResult) ) {
  warning("Poor numeric results with Anouk test file")
} else {
  print("Results are OK with Anouk test file")
}
```

---

knowndbs *Description of databases known by an ACNUC server*

Description

Returns, for each database known by the server, its name (a valid value for the `bank` argument of `choosebank`), availability (off means temporarily unavailable), and description.

Usage

```
knowndbs(tag = c(NA, "TP", "TEST", "DEV"), socket = autosocket())
kdb(tag = c(NA, "TP", "TEST", "DEV"), socket = autosocket())
```

Arguments

`tag`	default to NA, see details
`socket`	an object of class `sockconn` connecting to a remote ACNUC database (default is a socket to the last opened database).

Details

When the optional `tag` argument is used, only databases tagged with the given string are listed; when this argument is NA (by default), only untagged databases are listed. The `tag` argument thus allows to identify series of special purpose (tagged) databases, in addition to default (untagged) ones.

Value

A dataframe with 3 columns:

`bank`	string. Valid bank values known by the ACNUC server
`status`	string. "on" means available, "off" means temporarily unavailable
`info`	string. short description of the database

Author(s)

J.R. Lobry

References

http://pbil.univ-lyon1.fr/databases/acnuc.html

`citation("seqinr")`

The full list of untagged and tagged databases is here : http://pbil.univ-lyon1.fr/databases/acnuc/banques_raa.php.

See Also

choosebank when called without arguments.

Examples

```
## Not run:

  ### Need internet connection
  choosebank("emblTP")
  kdb()
  closebank()

## End(Not run)
```

---

lseqinr *To see what's inside the package seqinr*

Description

This is just a shortcut for ls("package:seqinr")

Usage

`lseqinr()`

Value

The list of objects in the package seqinr

Note

Use `library(help=seqinr)` to have a summary of the functionc available in the package.

Author(s)

J.R. Lobry

References

```
citation("seqinr")
```

Examples

```
lseqinr()
```

`m16j` *Fragment of the E. coli chromosome*

Description

A fragment of the *E. coli* chromosome that was used in Lobry (1996) to show the change in GC skew at the origin of replication (*i.e.* the chirochore structure of bacterial chromosomes)

Usage

```
data(m16j)
```

Format

A string of 1,616,539 characters

Details

The sequence used in Lobry (1996) was a 1,616,174 bp fragment obtained from the concatenation of nine overlapping sequences (U18997, U00039, L10328, M87049, L19201, U00006, U14003, D10483, D26562. Ambiguities have been resolved since then and its was a chimeric sequence from K-12 strains MG1655 and W3110, the sequence used here is from strain MG1655 only (Blattner *et al.* 1997).

Source

Escherichia coli K-12 strain MG1655. Fragment from U00096 from the EBI Genome Reviews. Acnuc Release 7. Last Updated: Feb 26, 2007. XX DT 18-FEB-2004 (Rel. .1, Created) DT 09-JAN-2007 (Rel. 65, Last updated, Version 70) XX

References

Lobry, J.R. (1996) Asymmetric substitution patterns in the two DNA strands of bacteria. *Molecular Biology and Evolution*, **13**:660-665.

F.R. Blattner, G. Plunkett III, C.A. Bloch, N.T. Perna, V. Burland, M. Rilley, J. Collado-Vides, J.D. Glasner, C.K. Rode, G.F. Mayhew, J. Gregor, N.W. Davis, H.A. Kirkpatrick, M.A. Goeden, D.J. Rose, B. Mau, and Y. Shao. (1997) The complete genome sequence of *Escherichia coli* K-12. *Science*, **277**:1453-1462

```
citation("seqinr")
```

Examples

```
#
# Load data:
#
data(m16j)
#
# Define a function to compute the GC skew:
#
gcskew <- function(x) {
  if (!is.character(x) || length(x) > 1)
  stop("single string expected")
  tmp <- tolower(s2c(x))
  nC <- sum(tmp == "c")
  nG <- sum(tmp == "g")
  if (nC + nG == 0)
  return(NA)
  return(100 * (nC - nG)/(nC + nG))
}
#
# Moving window along the sequence:
#
step <- 10000
wsize <- 10000
starts <- seq(from = 1, to = nchar(m16j), by = step)
starts <- starts[-length(starts)]
n <- length(starts)
result <- numeric(n)
for (i in seq_len(n)) {
  result[i] <- gcskew(substr(m16j, starts[i], starts[i] + wsize - 1))
}
#
# Plot the result:
#
xx <- starts/1000
yy <- result
n <- length(result)
hline <- 0
plot(yy ~ xx, type = "n", axes = FALSE, ann = FALSE, ylim = c(-10, 10))
polygon(c(xx[1], xx, xx[n]), c(min(yy), yy, min(yy)), col = "black", border = NA)
usr <- par("usr")
rect(usr[1], usr[3], usr[2], hline, col = "white", border = NA)
lines(xx, yy)
abline(h = hline)
box()
axis(1, at = seq(0, 1600, by = 200))
axis(2, las = 1)
title(xlab = "position (Kbp)", ylab = "(C-G)/(C+G) [percent]", main = expression(paste("GC s
arrows(860, 5.5, 720, 0.5, length = 0.1, lwd = 2)
text(860, 5.5, "origin of replication", pos = 4)
```

`modifylist` *Modification of an ACNUC list*

Description

This function modifies a previously existing ACNUC list by selecting sequences either by length, either by date, either for the presence of a given string in annotations.

Usage

```
modifylist(listname, modlistname = listname, operation, type = c("length", "date",
```

Arguments

`listname`	the name of the ACNUC list to modify
`modlistname`	the name of the modified ACNUC list. Default is to use the same list name so that previous list is lost.
`operation`	a string of character describing the operation to be done, see details.
`type`	the type of operation, could be one of `"length", "date", "scan"`. Default is `"length"`
`socket`	an object of class `sockconn` connecting to a remote ACNUC database (default is a socket to the last opened database).
`virtual`	if TRUE, no attempt is made to retrieve the information about all the elements of the list. In this case, the `req` component of the list is set to NA.
`verbose`	logical, if TRUE mode verbose is on

Details

Example of possible values for the argument `operation`:

length as in "> 10000" or "< 500"

date as in "> 1/jul/2001" or "< 30/AUG/98"

scan specify the string to be searched for

Character < is to be understood as <= and > likewise.

Value

The result is directly assigned to the object `modlistname` in the user workspace. This is an objet of class `qaw`, a list with the following 6 components:

The result is directly assigned to the object `listname` in the user workspace. This is an objet of class `qaw`, a list with the following 6 components:

`call`	the original call
`name`	the ACNUC list name

nelem	the number of elements (for instance sequences) in the ACNUC list
typelist	the type of the elements of the list. Could be SQ for a list of sequence names, KW for a list of keywords, SP for a list of species names.
req	a list of sequence names that fit the required criteria or NA when called with parameter virtual is TRUE
socket	the socket connection that was used

Author(s)

J.R. Lobry

References

http://pbil.univ-lyon1.fr/databases/acnuc.html

```
citation("seqinr")
```

See Also

choosebank, query and prepgetannots to select the annotation lines for scan.

Examples

```
## Not run:  # Need internet connection
  choosebank("emblTP")
  query("mylist", "sp=felis catus et t=cds", virtual=TRUE)
  mylist$nelem # 603 sequences
  stopifnot(mylist$nelem == 603)

  # select sequences with at least 1000 bp:
  modifylist("mylist", operation = ">1000", virtual = TRUE)
  mylist$nelem # now, only 132 sequences
  stopifnot(mylist$nelem == 132)

  # scan for "felis" in annotations:
  modifylist("mylist", op = "felis", type = "scan", virtual = TRUE)
  mylist$nelem # now, only 33 sequences
  stopifnot(mylist$nelem == 33)

  # modify by date:
  modifylist("mylist", op = "> 1/jul/2001", type = "date", virtual = TRUE)
  mylist$nelem # now, only 15 sequences
  stopifnot(mylist$nelem == 15)

  # Summary of current ACNUC lists, one list called MYLIST on sever:
  sapply(alr()$rank, getliststate)
  closebank()
  ## End(Not run)
```

n2s *function to convert the numeric encoding of a DNA sequence into a vector of characters*

Description

By default, if no 'levels' arguments is provided, this function will just transform your vector of integer into a DNA sequence according to the lexical order: `0 -> "a"`, `1 -> "c"`, `2 -> "g"`, `3 -> "t"`, `others -> NA`.

Usage

```
n2s(nseq, levels = c("a", "c", "g", "t"), base4 = TRUE)
```

Arguments

`nseq`	A vector of integers
`levels`	the translation vector
`base4`	when this logical is true, the numerical encoding of `levels` starts at 0, when it is false the numerical encoding of `levels` starts at 1.

Value

a vector of characters

Author(s)

J.R. Lobry

References

```
citation("seqinr")
```

See Also

s2n

Examples

```
##example of the default behaviour:
nseq <- sample(x = 0:3, size = 100, replace = TRUE)
n2s(nseq)
# Show what happens with out-of-range and NA values:
nseq[1] <- NA
nseq[2] <- 777
n2s(nseq)[1:10]
# How to get an RNA instead:
n2s(nseq, levels = c("a", "c", "g", "u"))
```

oriloc *Prediction of origin and terminus of replication in bacteria.*

Description

This program finds the putative origin and terminus of replication in procaryotic genomes. The program discriminates between codon positions.

Usage

```
oriloc(seq.fasta = system.file("sequences/ct.fasta", package ="seqinr"),
 g2.coord = system.file("sequences/ct.predict", package = "seqinr"),
 glimmer.version = 3,
oldoriloc = FALSE, gbk = NULL, clean.tmp.files = TRUE, rot = 0)
```

Arguments

seq.fasta	Character: the name of a file which contains the DNA sequence of a bacterial chromosome in fasta format. The default value, `system.file("sequences/ct.fasta", package ="seqinr")`, is to use the fasta file `ct.fasta` which is distributed in the `sequences` folder in the seqinR package. This is the file for the complete genome sequence of *Chlamydia trachomatis* that was used in Frank and Lobry (2000). You can replace this by something like `seq.fasta = "myseq.fasta"` to work with your own data if the file `myseq.fasta` is present in the current working directory (see `getwd`), or give a full path access to the sequence file (see `file.choose`).
g2.coord	Character: the name of file which contains the output of glimmer program (`*.predict` in glimmer version 3)
glimmer.version	Numeric: glimmer version used, could be 2 or 3
oldoriloc	Logical: to be set at TRUE to reproduce the (deprecated) outputs of previous (publication date: 2000) version of the oriloc program.
gbk	Character: the URL of a file in GenBank format. When provided `oriloc` use as input a single GenBank file instead of the `seq.fasta` and the `g2.coord`. A local temporary copy of the GenBank file is made with `download.file` if `gbk` starts with `http://` or `ftp://` or `file://` and whith `file.copy` otherwise. The local copy is then used as input for `gb2fasta` and `gbk2g2` to produce a fasta file and a glimmer-like (version 2) file, respectively, to be used by oriloc instead of `seq.fasta` and `g2.coord`.
clean.tmp.files	Logical: if TRUE temporary files generated when working with a GenBank file are removed.
rot	Integer, with zero default value, used to permute circularly the genome.

Details

The method builds on the fact that there are compositional asymmetries between the leading and the lagging strand for replication. The programs works only with third codon positions so as to increase the signal/noise ratio. To discriminate between codon positions, the program use as input either an annotated genbank file, either a fasta file and a glimmer2.0 (or glimmer3.0) output file.

Value

A data.frame with seven columns: `g2num` for the CDS number in the `g2.coord` file, `start.kb` for the start position of CDS expressed in Kb (this is the position of the first occurence of a nucleotide in a CDS *regardless* of its orientation), `end.kb` for the last position of a CDS, `CDS.excess` for the DNA walk for gene orientation (+1 for a CDS in the direct strand, -1 for a CDS in the reverse strand) cummulated over genes, `skew` for the cummulated composite skew in third codon positions, `x` for the cummulated T - A skew in third codon position, `y` for the cummulated C - G skew in third codon positions.

Note

The method works only for genomes having a single origin of replication from which the replication is bidirectional. To detect the composition changes, a DNA-walk is performed. In a 2-dimensional DNA walk, a C in the sequence corresponds to the movement in the positive y-direction and G to a movement in the negative y-direction. T and A are mapped by analogous steps along the x-axis. When there is a strand asymmetry, this will form a trajectory that turns at the origin and terminus of replication. Each step is the sum of nucleotides in a gene in third codon positions. Then orthogonal regression is used to find a line through this trajectory. Each point in the trajectory will have a corresponding point on the line, and the coordinates of each are calculated. Thereafter, the distances from each of these points to the origin (of the plane), are calculated. These distances will represent a form of cumulative skew. This permets us to make a plot with the gene position (gene number, start or end position) on the x-axis and the cumulative skew (distance) at the y-axis. Depending on where the sequence starts, such a plot will display one or two peaks. Positive peak means origin, and negative means terminus. In the case of only one peak, the sequence starts at the origin or terminus site.

Author(s)

J.R. Lobry and A.C. Frank

References

More illustrated explanations to help understand oriloc outputs are available there: http://pbil.univ-lyon1.fr/software/Oriloc/howto.html.

Examples of oriloc outputs on real sequence data are there: http://pbil.univ-lyon1.fr/software/Oriloc/index.html.

The original paper for oriloc:
Frank, A.C., Lobry, J.R. (2000) Oriloc: prediction of replication boundaries in unannotated bacterial chromosomes. *Bioinformatics*, **16**:566-567.

http://bioinformatics.oupjournals.org/cgi/reprint/16/6/560

A simple informal introduction to DNA-walks:
Lobry, J.R. (1999) Genomic landscapes. *Microbiology Today*, **26**:164-165.
http://www.socgenmicrobiol.org.uk/QUA/049906.pdf

An early and somewhat historical application of DNA-walks:
Lobry, J.R. (1996) A simple vectorial representation of DNA sequences for the detection of replication origins in bacteria. *Biochimie*, **78**:323-326.

Glimmer, a very efficient open source software for the prediction of CDS from scratch in prokaryotic genome, is decribed at http://www.cbcb.umd.edu/software/glimmer/.
For a description of Glimmer 1.0 and 2.0 see:

Delcher, A.L., Harmon, D., Kasif, S., White, O., Salzberg, S.L. (1999) Improved microbial gene identification with GLIMMER, *Nucleic Acids Research*, **27**:4636-4641.

Salzberg, S., Delcher, A., Kasif, S., White, O. (1998) Microbial gene identification using interpolated Markov models, *Nucleic Acids Research*, **26**:544-548.

```
citation("seqinr")
```

See Also

draw.oriloc, rearranged.oriloc

Examples

```
## Not run:
#
# A little bit too long for routine checks because oriloc() is already
# called in draw.oriloc.Rd documentation file. Try example(draw.oriloc)
# instead, or copy/paste the following code:
#
out <- oriloc()
plot(out$st, out$sk, type = "l", xlab = "Map position in Kb",
    ylab = "Cumulated composite skew",
    main = expression(italic(Chlamydia~~trachomatis)~~complete~~genome))
#
# Example with a single GenBank file:
#
out2 <- oriloc(gbk=system.file("sequences/ct.gbk", package = "seqinr"))
draw.oriloc(out2)
#
# (some warnings are generated because of join in features and a gene that
# wrap around the genome)
#
```

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

---

pK *pK values for the side chain of charged amino acids from various sources*

Description

This compilation of pK values is from Joanna Kiraga (2008).

Usage

```
data(pK)
```

Format

A data frame with the seven charged amino-acid in row and six sources in column. The rownames are the one-letter code for amino-acids.

Source

Table 2 in Kiraga (2008).

References

Kiraga, J. (2008) Analysis and computer simulations of variability of isoelectric point of proteins in the proteomes. PhD thesis, University of Wroclaw, Poland.

Bjellqvist, B., Hughes, G.J., Pasquali, Ch., Paquet, N., Ravier, F., Sanchez, J.Ch., Frutige,r S., Hochstrasser D. (1993) The focusing positions of polypeptides in immobilized pH gradients can be predicted from their amino acid sequences. *Electrophoresis*, **14**:1023-1031.

EMBOSS data are from http://emboss.sourceforge.net/apps/release/5.0/emboss/apps/iep.html.

Murray, R.K., Granner, D.K., Rodwell, V.W. (2006) *Harper's illustrated Biochemistry.* 27th edition. Published by The McGraw-Hill Companies.

Sillero, A., Maldonado, A. (2006) Isoelectric point determination of proteins and other macromolecules: oscillating method. *Comput Biol Med.*, **36**:157-166.

Solomon, T.W.G. (1998) *Fundamentals of Organic Chemistry*, 5th edition. Published by Wiley.

Stryer L. (1999) *Biochemia*. czwarta edycja. Wydawnictwo Naukowe PWN.

```
citation("seqinr")
```

Examples

```
data(pK)
data(SEQINR.UTIL) # for N and C terminal pK values
prot <- s2c("ACDEFGHIKLMNPQRSTVWY")
compoAA <- table(factor(prot, levels = LETTERS))
nTermR <- which(LETTERS == prot[1])
cTermR <- which(LETTERS == prot[length(seq)])

computeCharge <- function(pH, compoAA, pK, nTermResidue, cTermResidue){
  cter <- 10^(-SEQINR.UTIL$pk[cTermResidue,1]) /
     (10^(-SEQINR.UTIL$pk[cTermResidue,1]) + 10^(-pH))
  nter <- 10^(-pH) / (10^(-SEQINR.UTIL$pk[nTermResidue,2]) + 10^(-pH))
  carg <- as.vector(compoAA['R'] * 10^(-pH) / (10^(-pK['R']) + 10^(-pH)))
  chis <- as.vector(compoAA['H'] * 10^(-pH) / (10^(-pK['H']) + 10^(-pH)))
  clys <- as.vector(compoAA['K'] * 10^(-pH) / (10^(-pK['K']) + 10^(-pH)))
  casp <- as.vector(compoAA['D'] * 10^(-pK['D']) /(10^(-pK['D']) + 10^(-pH)))
  cglu <- as.vector(compoAA['E'] * 10^(-pK['E']) / (10^(-pK['E']) + 10^(-pH)))
  ccys <- as.vector(compoAA['C'] * 10^(-pK['C']) / (10^(-pK['C']) + 10^(-pH)))
  ctyr <- as.vector(compoAA['Y'] * 10^(-pK['Y']) / (10^(-pK['Y']) + 10^(-pH)))
  charge <- carg + clys + chis + nter - (casp + cglu + ctyr + ccys + cter)
  return(charge)
}

pHseq <- seq(from = 0, to = 14, by = 0.1)
Bje <- pK$Bjellqvist
names(Bje) <- rownames(pK)
res <- computeCharge(pHseq, compoAA, Bje, nTermR, cTermR)
plot(pHseq, res, type = "l", ylab = "Charge", las = 1,
  main = paste("Charge of protein\n",c2s(prot)),
  xlab = "pH")
for(j in 2:ncol(pK)){
  src <- pK[,j]
  names(src) <- rownames(pK)
  res <- computeCharge(pHseq, compoAA, src, nTermR, cTermR)
  lines(pHseq, res, lty = j, col = rainbow(5)[j])
}

abline(h=0)
abline(v=computePI(prot))
legend("bottomleft", inset = 0.01, colnames(pK), lty = 1:6, col = c("black", rainbow(5)))
```

---

parser.socket *Utility function to parse answers from an ACNUC server*

Description

Answers from server looks like : "code=0&lrank=2&count=150513&type=SQ&locus=F".

Usage

```
parser.socket(onelinefromserver, verbose = FALSE)
```

Arguments

onelinefromserver	a string
verbose	logical, if TRUE mode verbose is on

Value

A vector of mode character or NULL if onelinefromserver is NULL or if its length is 0.

Author(s)

J.R. Lobry

References

```
citation("seqinr")
```

See Also

choosebank, query

Examples

```
stopifnot(all(parser.socket("code=0&lrank=2&count=150513&type=SQ&locus=F")
                                        == c("0", "2", "150513", "SQ", "F")))
```

---

permutation *Sequence permutation according to several different models*

Description

Generates a random permutation of a given sequence, according to a given model. Available models are : base, position, codon, syncodon.

Usage

```
permutation(sequence,modele='base',frame=0,replace=FALSE,prot=FALSE,numcode=1,ucowe
```

Arguments

sequence	A nucleic acids sequence
modele	A string of characters describing the model chosen for the random generation
frame	Only active for the position, codon, syncodon models: starting position of CDS as in splitseq
replace	This option is not active for the syncodon model: if TRUE, sampling is done with replacement

prot	Only available for the codon model: if TRUE, the first and last codons are preserved, and only intern codons are shuffled
numcode	Only available for the syncodon model: the genetic code number as in translate.
ucoweight	A list of weights containing the desired codon usage bias as generated by ucoweight. If none is specified, the codon usage of the given sequence is used.

Details

The base model allows for random sequence generation by shuffling (with/without replacement) of all bases in the sequence.

The position model allows for random sequence generation by shuffling (with/without replacement) of bases within their position in the codon (bases in position I, II or III stay in position I, II or III in the new sequence.

The codon model allows for random sequence generation by shuffling (with/without replacement) of codons.

The syncodon model allows for random sequence generation by shuffling (with/without replacement) of synonymous codons.

Value

a sequence generated from the original one by a given model

Author(s)

Leonor Palmeira

References

```
citation("seqinr")
```

See Also

synsequence

Examples

```
data(ec999)
sequence=ec999[1][[1]]

new=permutation(sequence,modele='base')
identical(all.equal(count(new,1),count(sequence,1)),TRUE)

new=permutation(sequence,modele='position')
identical(all.equal(GC(new),GC(sequence)),TRUE)
identical(all.equal(GC2(new),GC2(sequence)),TRUE)
identical(all.equal(GC3(new),GC3(sequence)),TRUE)

new=permutation(sequence,modele='codon')
identical(all.equal(uco(new),uco(sequence)),TRUE)
```

```
new=permutation(sequence,modele='syncodon',numcode=1)
identical(all.equal(translate(new),translate(sequence)),TRUE)
```

---

plot.SeqAcnucWeb *To Plot Subsequences on the Parent Sequence*

Description

This function plots all the type of subsequences on a parent sequence. Subsequences are represented by colored rectangle on the parent sequence. For example, types could be CDS, TRNA, RRNA .... In order to get all the types that are available for the selected database, use `getType`.

Usage

```
## S3 method for class 'SeqAcnucWeb':
plot(x, types = getType()$sname, socket = autosocket(), ...)
```

Arguments

`x`	A sequence of class `SeqAcnucWeb`
`types`	The type of subsequences to plot. Default value is to consider all possible subsequence types.
`socket`	an object of class `sockconn` connecting to a remote ACNUC database (default is a socket to the last opened database).
`...`	not currently used

Value

An invisible list giving, for each subsequence, its position on the parent sequence.

Author(s)

D. Charif and J.R. Lobry

References

http://pbil.univ-lyon1.fr/databases/acnuc.html

```
citation("seqinr")
```

See Also

`getType`, `query`

Examples

```
## Not run:

  ### Need internet connection
  choosebank("hovernucl")
  query("list", "AC=AB000425")
  plot(list$req[[1]])

## End(Not run)
```

---

pmw *Protein Molecular Weight*

Description

With default parameter values, returns the apparent molecular weight of one mole (6.0221415 e+23) of the input protein expressed in gram at see level on Earth with terrestrial isotopic composition.

Usage

```
pmw(seqaa, Ar = c(C = 12.0107, H = 1.00794, O = 15.9994,
N = 14.0067, P = 30.973762, S = 32.065), gravity = 9.81,
unit = "gram", checkseqaa = TRUE)
```

Arguments

seqaa	a protein sequence as a vector of single chars. Allowed values are "*ACDEFGHIKLMNPQRSTVWY", non allowed values are ignored.
Ar	a named vector for the mean relative atomic masses of CHONPS atoms. Defaults values are from to the natural terrestrial sources according to the 43rd IUPAC General Assembly in Beijing, China in August 2005 (See http://www.iupac.org/reports/periodic_table/ for updates).
gravity	gravitational field constant in standard units. Defaults to 9.81 m/s2, that is to the average value at see level on Earth. Negative values are not allowed.
unit	a string that could be "gram" to get the result in grams (1 g = 0.001 kg) or "N" to get the result in Newton units (1 N = 1 kg.m/s2).
checkseqaa	if TRUE pmw() warns if a non-allowed character in seqaa is found.

Details

Algorithm Computing the molecular mass of a protein is close to a linear form on amino-acid frequencies, but not exactly since we have to remove n - 1 water molecules for peptidic bound formation.

Cysteine All cysteines are supposed to be in reduced (-SH) form.

Methionine All methionines are supposed to be not oxidized.

Modifications No post-traductional modifications (such as phosphorylations) are taken into account.

Rare Rare amino-acids (pyrolysine and selenocysteine) are not handled.

Warning Do not use defaults values for Ar to compute the molecular mass of alien's proteins: the isotopic composition for CHONPS atoms could be different from terrestrial data in a xenobiotic context. Some aliens are easily offended, make sure not to initiate one more galactic war by repporting wrong results.

Value

The protein molecular weight as a single numeric value.

Author(s)

J.R. Lobry

References

```
citation("seqinr")
```

See Also

s2c, c2s, aaa, a

Examples

```
allowed <- s2c("*ACDEFGHIKLMNPQRSTVWY") # All allowed chars in a protein
pmw(allowed)
all.equal(pmw(allowed), 2395.71366) # Should be true on most platforms
#
# Compute the apparent molecular weight on Moon surface:
#
pmw(allowed, g = 1.6)
#
# Compute the apparent molecular weight in absence of gravity:
#
pmw(allowed, g = 0) # should be zero
#
# Reports results in Newton units:
#
pmw(allowed, unit = "N")
#
# Compute the mass in kg of one mol of this protein:
#
pmw(allowed)/10^3
#
# Compute the mass for all amino-acids:
#
sapply(allowed[-1], pmw) -> aamw
names(aamw) <- aaa(names(aamw))
aamw
```

`prepgetannots` *Select annotation lines in an ACNUC database*

Description

This function is called before using `getAnnot` or `modifylist` with a `scan` type operation to select the annotation lines to be returned or scanned.

Usage

```
prepgetannots(what = "all", setfor = c("scan", "getannots"),
                        socket = autosocket(), verbose = FALSE)
pga(what = "all", setfor = c("scan", "getannots"),
                        socket = autosocket(), verbose = FALSE)
```

Arguments

`what`	the default "all" means that all annotation lines are selected. This can be more specific, see details.
`setfor`	this is used when `what` has its default "all" value. The behaviour is different for `getAnnot` and `modifylist` with a `scan` type operation: annotations but not sequences are scanned, but sequences can be returned by `getAnnot`. The default value is "scan".
`socket`	an object of class `sockconn` connecting to an ACNUC server
`verbose`	logical, if TRUE mode verbose is on

Details

The names of annotation lines in the opened ACNUC database is returned by `countfreelists`, they are forced to upper case letters by `prepgetannots` when supplied with the `what` argument.

For the EMBL/SWISSPROT format, keys are: ALL, AC, DT, KW, OS, OC, OG, OH, RN, RC, RP, RX, RA, RG, RT, RL, DR, AH, AS, CC, FH, FT, SQ, SEQ.

For GenBank: ALL, ACCESSION, VERSION, KEYWORDS, SOURCE, ORGANISM, REFERENCE, AUTHORS, CONSRTM, TITLE, JOURNAL, PUBMED, REMARK, COMMENT, FEATURES, ORIGIN, SEQUENCE.

For FT (embl, swissprot) and FEATURES (GenBank), one or more specific feature keys can be specified using lines with only uppercase and such as

FEATURES|CDS FT|TRNA

Keys ALL and SEQ/SEQUENCE stand for all annotation and sequence lines, respectively. For the scan operation, key ALL stand for the DE/DEFINITION lines, and SEQ/SEQUENCE cannot be used (annotations but not sequence are scanned).

Value

The function returns invisibly the annotation lines names.

Author(s)

J.R. Lobry

References

```
citation("seqinr")
```

See Also

getAnnot, modifylist, countfreelists

Examples

```
 ## Not run:
# Need internet connection
  choosebank("genbank")
  query("mylist","n=AQF16SRRN")
  pga() # We want to scan all annotations, including FEATURES
  modifylist("mylist", operation = "strain", type = "scan")
  mylist$nelem # should be 1

## End(Not run)
```

---

prettyseq *Text representation of a sequence from an ACNUC server*

Description

To get a text representation of sequence of rank `num` and of its subsequences, with `bpl` bases per line (default = 60), and with optional translation of protein-coding subsequences

Usage

```
prettyseq(num, bpl = 60, translate = TRUE, socket = autosocket())
```

Arguments

`num`	rank of the sequence in the ACNUC database
`bpl`	number of base per line
`translate`	should coding sequences be translated?
`socket`	an object of class `sockconn` connecting to a remote ACNUC database (default is a socket to the last opened database).

Value

An invisible vector of string. The output is redirected to the console.

Author(s)

J.R. Lobry

References

http://pbil.univ-lyon1.fr/databases/acnuc.html

```
citation("seqinr")
```

See Also

choosebank, query

Examples

```
## Not run:

  ### Need internet connection
  choosebank("emblTP")
  prettyseq(111)

## End(Not run)
```

---

print.SeqAcnucWeb *Print method for objects from class SeqAcnucWeb*

Description

Print the name, length, frame and genetic code number.

Usage

```
## S3 method for class 'SeqAcnucWeb':
print(x, ...)
```

Arguments

x	A sequence of class SeqAcnucWeb
...	Arguments passed to print

Value

None.

Author(s)

J.R. Lobry

References

```
citation("seqinr")
```

See Also

print

Examples

```
## Not run:

  ### Need internet connection
  choosebank("emblTP")
  query("mylist", "sp=felis catus")
  mylist$req[[1]]
  #      name   length    frame   ncbicg
  # "A06937"     "34"      "0"      "1"

## End(Not run)
```

`print.qaw` *Print method for objects from class qaw*

Description

Print the number of elements, their type and the corresponding query.

Usage

```
## S3 method for class 'qaw':
print(x, ...)
```

Arguments

`x`	A objet of class `qaw`
`...`	not used

Value

None.

Author(s)

J.R. Lobry

References

```
citation("seqinr")
```

See Also

print

Examples

```
## Not run:

  ### Need internet connection
  choosebank("emblTP")
  query("sp=felis catus")
  list1
  # 4732 SQ for sp=felis catus

## End(Not run)
```

---

prochlo *Zscore on three strains of Prochlorococcus marinus*

Description

This dataset contains the zscores computed with the codon model on all CDS from 3 strains of Procholorococcus marinus (as retrieved from Genome Reviews database on June 16, 2005)

Usage

```
data(prochlo)
```

Format

List of three dataframes of the zscore of each of the 16 dinucleotides on each CDS retrieved from the specific strain.

BX548174 strain adapted to living at a depth of 5 meters (high levels of UV exposure) `base` model on each intergenic sequence

AE017126 strain adapted to living at a depth of 120 meters (low levels of UV exposure)

BX548175 strain adapted to living at a depth of 135 meters (low levels of UV exposure)

References

Palmeira, L., Guéguen, L. and Lobry JR. (2006) UV-targeted dinucleotides are not depleted in light-exposed Prokaryotic genomes. *Molecular Biology and Evolution*, **23**:2214-2219.
http://mbe.oxfordjournals.org/cgi/reprint/23/11/2214

```
citation("seqinr")
```

See Also

zscore

Examples

```
#
# Show the four YpY for the three ecotypes:
#
data(prochlo)
oneplot <- function(x){
  plot(density(prochlo$BX548174[, x]),
    ylim = c(0,0.4), xlim = c(-4,4), lty=3,
    main = paste(substr(x,1,1), "p", substr(x,2,2), " bias", sep = ""),
    xlab="",ylab="",las=1, type = "n")
  rect(-10,-1,-1.96,10, col = "yellow", border = "yellow")
  rect(1.96,-1,10,10, col = "yellow", border = "yellow")
  lines(density(prochlo$BX548174[, x]),lty=3)
  lines(density(prochlo$AE017126[, x]),lty=2)
  lines(density(prochlo$BX548175[, x]),lty=1)
  abline(v=c(-1.96,1.96),lty=5)
  box()
}
par(mfrow=c(2,2),mar=c(2,3,2,0.5) + 0.1)
oneplot("CT")
oneplot("TC")
oneplot("CC")
oneplot("TT")
#
# Show YpY biases with respect to light exposure
#
curdev <- getOption("device")
OK <- FALSE
devlist <- c("X11", "windows", "quartz") # interactive with width and height in inches
for(i in devlist){
  if(exists(i) && identical(get(i), curdev)){
    OK <- TRUE
    break
  }
}
if(OK){
  curdev(width = 18, height = 11)
  par(oma = c(0, 0, 3, 0), mfrow = c(1, 2), mar = c(5, 4, 0, 0), cex = 1.5)
  example(waterabs, ask = FALSE) #left figure

  par(mar = c(5, 0, 0, 2))
  plot(seq(-5, 3, by = 1), seq(0, 150, length = 9), col = "white",
    ann = FALSE, axes = FALSE, xaxs = "i", yaxs = "i")
  axis(1, at = c(-1.96, 0, 1.96), labels = c(-1.96, 0, 1.96))
  lines(rep(-1.96, 2),c(0, 150),lty=2)
  lines(rep(1.96, 2), c(0, 150),lty=2)
  title(xlab = "zscore distribution", cex = 1.5, adj = 0.65)
```

```
selcol <- c(6, 8, 14, 16)
z5 <- prochlo$BX548174[, selcol]
z120 <- prochlo$AE017126[, selcol]
z135 <- prochlo$BX548175[, selcol]

todo <- function(who, xx, col = "black", bottom, loupe){
      dst <- density(who[, xx])
      sel <- which(dst$x >= -3)
      lines(dst$x[sel], dst$y[sel]*loupe + (bottom), col = col)
}
todo2 <- function(who, bottom, loupe){
  todo(who, "CC", "blue", bottom, loupe)
  todo(who, "CT", "red", bottom, loupe)
  todo(who, "TC", "green", bottom, loupe)
  todo(who, "TT", "black", bottom, loupe)
}
todo3 <- function(bottom, who, leg, loupe = 90){
  lines(c(-5,-3), c(150 - leg, bottom + 20))
  rect(-3,bottom,3,bottom+40)
  text(-2.6,bottom+38, paste(leg, "m"))
  todo2(who, bottom, loupe)
}

todo3(bottom = 110, who = z5, leg = 5)
todo3(bottom = 50, who = z120, leg = 120)
todo3(bottom = 5, who = z135, leg = 135)

legend(-4.5,110,c('CpC','CpT','TpC','TpT'),lty=1,pt.cex=cex,
  col=c('blue','red','green','black'))

mtext(expression(paste("Dinucleotide composition for three ",
  italic("Prochlorococcus marinus")," ecotypes")), outer = TRUE, cex = 2, line = 1)
}
```

`query` *To get a list of sequence names from an ACNUC data base located on the web*

Description

This is a major command of the package. It executes all sequence retrievals using any selection criteria the data base allows. The sequences are coming from ACNUC data base located on the web and they are transfered by socket. The command produces the list of all sequence names that fit the required criteria. The sequence names belong to the class of sequence `SeqAcnucWeb`.

Usage

```
query(listname, query, socket = autosocket(), invisible = TRUE, verbose = FALSE, vi
```

Arguments

`listname`	The name of the list as a quoted string of chars
`query`	A quoted string of chars containing the request with the syntax given in the details section
`socket`	an object of class `sockconn` connecting to a remote ACNUC database (default is a socket to the last opened database).
`invisible`	if `FALSE`, the result is returned visibly.
`verbose`	if `TRUE`, verbose mode is on
`virtual`	if `TRUE`, no attempt is made to retrieve the information about all the elements of the list. In this case, the `req` component of the list is set to `NA`.

Details

The query language defines several selection criteria and operations between lists of elements matching criteria. It creates mainly lists of sequences, but also lists of species (or, more generally, taxa) and of keywords. See `http://pbil.univ-lyon1.fr/databases/acnuc/cfonctions.html#QUERYLANGUAGE` for the last update of the description of the query language.

Selection criteria (no space before the = sign) are:

SP=taxon seqs attached to taxon or any other below in tree; @ wildcard possible

TID=id seqs attached to given numerical NCBI's taxon id

K=keyword seqs attached to keyword or any other below in tree; @ wildcard possible

T=type seqs of specified type

J=journalname seqs published in journal specified using defined journal code

R=refcode seqs from reference specified such as in jcode/volume/page (e.g., JMB/13/5432)

AU=name seqs from references having specified author (only last name, no initial)

AC=accessionno seqs attached to specified accession number

N=seqname seqs of given name (ID or LOCUS); @ wildcard possible

Y=year seqs published in specified year; > and < can be used instead of =

O=organelle seqs from specified organelle named following defined code (e.g., chloroplast)

M=molecule seqs from specified molecule as named in ID or LOCUS annotation records

ST=status seqs from specified data class (EMBL) or review level (UniProt)

F=filename seqs whose names are in given file, one name per line (unimplemented use `clfcd` instead)

FA=filename seqs attached to accession numbers in given file, one number per line (unimplemented use `clfcd` instead)

FK=filename produces the list of keywords named in given file, one keyword per line (unimplemented use `clfcd` instead)

FS=filename produces the list of species named in given file, one species per line (unimplemented use `clfcd` instead)

listname the named list that must have been previously constructed

Operators (always followed and preceded by blanks or parentheses) are:

AND intersection of the 2 list operands

OR union of the 2 list operands

NOT complementation of the single list operand

PAR compute the list of parent seqs of members of the single list operand

SUB add subsequences of members of the single list operand

PS project to species: list of species attached to member sequences of the operand list

PK project to keywords: list of keywords attached to member sequences of the operand list

UN unproject: list of seqs attached to members of the species or keywords list operand

SD compute the list of species placed in the tree below the members of the species list operand

KD compute the list of keywords placed in the tree below the members of the keywords list operand

The query language is case insensitive.Three operators (AND, OR, NOT) can be ambiguous because they can also occur within valid criterion values. Such ambiguities can be solved by encapsulating elementary selection criteria between escaped double quotes.

Value

The result is directly assigned to the object `listname` in the user workspace. This is an objet of class `qaw`, a list with the following 6 components:

`call`	the original call
`name`	the ACNUC list name
`nelem`	the number of elements (for instance sequences) in the ACNUC list
`typelist`	the type of the elements of the list. Could be SQ for a list of sequence names, KW for a list of keywords, SP for a list of species names.
`req`	a list of sequence names that fit the required criteria or `NA` when called with parameter `virtual` is `TRUE`
`socket`	the socket connection that was used

Note

Most of the documentation was imported from ACNUC help files written by Manolo Gouy

Author(s)

J.R. Lobry & D. Charif

References

To get the release date and content of all the databases located at the pbil, please look at the following url: http://pbil.univ-lyon1.fr/search/releases.php
Gouy, M., Milleret, F., Mugnier, C., Jacobzone, M., Gautier,C. (1984) ACNUC: a nucleic acid sequence data base and analysis system. *Nucl. Acids Res.*, **12**:121-127.
Gouy, M., Gautier, C., Attimonelli, M., Lanave, C., Di Paola, G. (1985) ACNUC - a portable retrieval system for nucleic acid sequence databases: logical and physical designs and usage. *Comput. Appl. Biosci.*, **3**:167-172.
Gouy, M., Gautier, C., Milleret, F. (1985) System analysis and nucleic acid sequence banks. *Biochimie*, **67**:433-436.

```
citation("seqinr")
```

See Also

choosebank, getSequence, getName, crelistfromclientdata

Examples

```
## Not run:
# Need internet connection
choosebank("genbank")
query("bb", "sp=Borrelia burgdorferi")
# To get the names of the 4 first sequences:
sapply(bb$req[1:4], getName)
# To get the 4 first sequences:
sapply(bb$req[1:4], getSequence, as.string = TRUE)

## End(Not run)
```

---

read.alignment *Read aligned sequence files in mase, clustal, phylip, fasta or msf format*

Description

Read a file in mase, clustal, phylip, fasta or msf format. These formats are used to store nucleotide or protein multiple alignments.

Usage

```
read.alignment(file, format, forceToLower = TRUE, File = NULL)
```

Arguments

file	the name of the file which the aligned sequences are to be read from. If it does not contain an absolute or relative path, the file name is relative to the current working directory, getwd.
format	a character string specifying the format of the file : mase, clustal, phylip, fasta or msf
forceToLower	a logical defaulting to TRUE stating whether the returned characters in the sequence should be in lower case (introduced in seqinR release 1.1-3).
File	synonymous of file maintained for backward compatibility. As from seqinR 1.1-3 this argument is deprecated and a warning is issued.

Details

"mase" The mase format is used to store nucleotide or protein multiple alignments. The beginning of the file must contain a header containing at least one line (but the content of this header may be empty). The header lines must begin by ;;. The body of the file has the following structure: First, each entry must begin by one (or more) commentary line. Commentary lines begin by the character ;. Again, this commentary line may be empty. After the commentaries, the name of the sequence is written on a separate line. At last, the sequence itself is written on the following lines.

"clustal" The CLUSTAL format (*.aln) is the format of the ClustalW multialignment tool output. It can be described as follows. The word CLUSTAL is on the first line of the file. The alignment is displayed in blocks of a fixed length, each line in the block corresponding to one sequence. Each line of each block starts with the sequence name (maximum of 10 characters), followed by at least one space character. The sequence is then displayed in upper or lower cases, '-' denotes gaps. The residue number may be displayed at the end of the first line of each block.

"msf" MSF is the multiple sequence alignment format of the GCG sequence analysis package. It begins with the line (all uppercase) !!NA_MULTIPLE_ALIGNMENT 1.0 for nucleic acid sequences or !!AA_MULTIPLE_ALIGNMENT 1.0 for amino acid sequences. Do not edit or delete the file type if its present.(optional). A description line which contains informative text describing what is in the file. You can add this information to the top of the MSF file using a text editor.(optional) A dividing line which contains the number of bases or residues in the sequence, when the file was created, and importantly, two dots (..) which act as a divider between the descriptive information and the following sequence information.(required) msf files contain some other information: the Name/Weight, a Separating Line which must include two slashes (//) to divide the name/weight information from the sequence alignment.(required) and the multiple sequence alignment.

"phylip" PHYLIP is a tree construction program. The format is as follows: the number of sequences and their length (in characters) is on the first line of the file. The alignment is displayed in an interleaved or sequential format. The sequence names are limited to 10 characters and may contain blanks.

"fasta" Sequence in fasta format begins with a single-line description (distinguished by a greater-than (>) symbol), followed by sequence data on the next line.

Value

An object of class `alignment` which is a list with the following components:

`nb`	the number of aligned sequences
`nam`	a vector of strings containing the names of the aligned sequences
`seq`	a vector of strings containing the aligned sequences
`com`	a vector of strings containing the commentaries for each sequence or `NA` if there are no comments

Author(s)

D. Charif, J.R. Lobry

References

```
citation("seqinr")
```

See Also

To read aligned sequences in NEXUS format, see the function `read.nexus` in the `CompPairWise` package. The NEXUS format is mainly used by the non-GPL commercial PAUP software.

Related functions: `as.matrix.alignment`, `read.fasta`, `write.fasta`, `reverse.align`, `dist.alignment`.

Examples

```
mase <- read.alignment(file = system.file("sequences/test.mase", package = "seqinr"), format
clustal <- read.alignment(file = system.file("sequences/test.aln", package = "seqinr"), form
phylip <- read.alignment(file = system.file("sequences/test.phylip", package = "seqinr"), fo
msf <- read.alignment(file = system.file("sequences/test.msf", package = "seqinr"), format =
fasta <- read.alignment(file = system.file("sequences/Anouk.fasta", package = "seqinr"), for
```

`read.fasta` *read FASTA formatted files*

Description

Read nucleic or amino-acid sequences from a file in FASTA format.

Usage

```
read.fasta(file = system.file("sequences/ct.fasta", package = "seqinr"),
  seqtype = c("DNA", "AA"), File = NULL, as.string = FALSE, forceDNAtolower = TRUE,
  set.attributes = TRUE, legacy.mode = TRUE, seqonly = FALSE, strip.desc = FALSE,
  bfa = FALSE, sizeof.longlong = .Machine$sizeof.longlong,
  endian = .Platform$endian, apply.mask = TRUE)
```

Arguments

file	The name of the file which the sequences in fasta format are to be read from. If it does not contain an absolute or relative path, the file name is relative to the current working directory, getwd. The default here is to read the ct.fasta file which is present in the sequences folder of the seqinR package.
seqtype	the nature of the sequence: DNA or AA, defaulting to DNA
File	Synonymous of file. As from seqinR >= 1.1-3 this argument is deprecated and a warning is issued
as.string	if TRUE sequences are returned as a string instead of a vector of single characters
forceDNAtolower	whether sequences with seqtype == "DNA" should be returned as lower case letters
set.attributes	whether sequence attributes should be set
legacy.mode	if TRUE lines starting with a semicolon ';' are ignored
seqonly	if TRUE, only sequences as returned without attempt to modify them or to get their names and annotations (execution time is divided approximately by a factor 3)
strip.desc	if TRUE the '>' at the beginning of the description lines is removed in the annotations of the sequences
bfa	logical. If TRUE the fasta file is in MAQ binary format (see details). Only for DNA sequences.
sizeof.longlong	the number of bytes in a C long long type. Only relevant for bfa = TRUE. See .Machine
endian	character string, "big" or "little", giving the endianness of the processor in use. Only relevant for bfa = TRUE. See .Platform
apply.mask	logical defaulting to TRUE. Only relevant for bfa = TRUE. When this flag is TRUE the mask in the MAQ binary format is used to replace non acgt characters in the sequence by the n character. For pure acgt sequences (without gaps or ambiguous bases) turning this to FALSE will save time.

Details

FASTA is a widely used format in biology, some FASTA files are distributed with the seqinr package, see the examples section below. Sequence in FASTA format begins with a single-line description (distinguished by a greater-than '>' symbol), followed by sequence data on the next lines. Lines starting by a semicolon ';' are ignored, as in the original FASTA program (Pearson and Lipman 1988). The sequence name is just after the '>' up to the next space ' ' character, trailling infos are ignored for the name but saved in the annotations.

The MAQ fasta binary format was introduced in seqinR 1.1-7 and has not been extensively tested. This format is used in the MAQ (Mapping and Assembly with Qualities) software (http://maq.sourceforge.net/). In this format the four nucleotides are coded with two bits and the sequence is stored as a vector of C unsigned long long. There is in addition a mask to locate non-acgt characters.

Value

By default `read.fasta` return a list of vector of chars. Each element is a sequence object of the class `SeqFastadna` or `SeqFastaAA`.

Author(s)

D. Charif, J.R. Lobry

References

Pearson, W.R. and Lipman, D.J. (1988) Improved tools for biological sequence comparison. *Proceedings of the National Academy of Sciences of the United States of America*, **85**:2444-2448

FIXME: a reference to MAQ when published.

`citation("seqinr")`

See Also

`write.fasta` to write sequences in a FASTA file, `gb2fasta` to convert a GenBank file into a FASTA file, `read.alignment` to read aligned sequences, `reverse.align` to get an alignment at the nucleic level from the one at the amino-acid level

Examples

```
#
# Simple sanity check with a small FASTA file:
#
  smallFastaFile <- system.file("sequences/smallAA.fasta", package = "seqinr")
  mySmallProtein <- read.fasta(file = smallFastaFile, as.string = TRUE, seqtype = "AA")[[1]]
  stopifnot(mySmallProtein == "SEQINRSEQINRSEQINRSEQINR*")
#
# Example of a DNA file in FASTA format:
#
  dnafile <- system.file("sequences/malM.fasta", package = "seqinr")
#
# Read with defaults arguments, looks like:
#
# $XYLEECOM.MALM
# [1] "a" "t" "g" "a" "a" "a" "a" "t" "g" "a" "a" "t" "a" "a" "a" "a" "g" "t"
# ...
  read.fasta(file = dnafile)
#
# The same but do not turn the sequence into a vector of single characters, looks like:
#
# $XYLEECOM.MALM
# [1] "atgaaaatgaataaaagtctcatcgtcctctgtttatcagcagggttactggcaagcgc
# ...
  read.fasta(file = dnafile, as.string = TRUE)
#
# The same but do not force lower case letters, looks like:
#
```

```
# $XYLEECOM.MALM
# [1] "ATGAAAATGAATAAAAGTCTCATCGTCCTCTGTTTATCAGCAGGGTTACTGGCAAGC
# ...
  read.fasta(file = dnafile, as.string = TRUE, forceDNAtolower = FALSE)
#
# Example of a protein file in FASTA format:
#
  aafile <- system.file("sequences/seqAA.fasta", package = "seqinr")
#
# Read the protein sequence file, looks like:
#
# $A06852
# [1] "M" "P" "R" "L" "F" "S" "Y" "L" "L" "G" "V" "W" "L" "L" "L" "S" "Q" "L"
# ...
  read.fasta(aafile, seqtype = "AA")
#
# The same, but as string and without attributes, looks like:
#
# $A06852
# [1] "MPRLFSYLLGVWLLLSQLPREIPGQSTNDFIKACGRELVRLWVEICGSVSWGRTALSLEEP
# QLETGPPAETMPSSITKDAEILKMMLEFVPNLPQELKATLSERQPSLRELQQSASKDSNLNFEEFK
# KIILNRQNEAEDKSLLELKNLGLDKHSRKKRLFRMTLSEKCCQVGCIRKDIARLC*"
#
  read.fasta(aafile, seqtype = "AA", as.string = TRUE, set.attributes = FALSE)
#
# Example with a FASTA file that contains comment lines starting with
# a semicolon character ';'
#
  legacyfile <- system.file("sequences/legacy.fasta", package = "seqinr")
  legacyseq <- read.fasta(file = legacyfile, as.string = TRUE)
  stopifnot( nchar(legacyseq) == 921 )
#
# Example of a MAQ binary fasta file produced with maq fasta2bfa ct.fasta ct.bfa
# on a platform where .Platform$endian == "little" and .Machine$sizeof.longlong == 8
#
  fastafile <- system.file("sequences/ct.fasta", package = "seqinr")
  bfafile <- system.file("sequences/ct.bfa", package = "seqinr")

  original <- read.fasta(fastafile, as.string = TRUE, set.att = FALSE)
  bfavers <- read.fasta(bfafile, as.string = TRUE, set.att = FALSE, bfa = TRUE,
    endian = "little", sizeof.longlong = 8)
  if(!identical(original, bfavers)){
     warning(paste("trouble reading bfa file with endian =", .Platform$endian,
    "and sizeof.longlong =", .Machine$sizeof.longlong))
  }
```

`readfirstrec` *Low level function to get the record count of the specified ACNUC index file*

Description

Called without arguments, the list of available values for argument type is returned.

Usage

```
readfirstrec(socket = autosocket(), type)
```

Arguments

socket	an object of class sockconn connecting to a remote ACNUC database (default is a socket to the last opened database).
type	the ACNUC index file

Details

Available index files are:

AUT AUTHOR one record for each author name (last name only, no initials)

BIB BIBLIO one record for each reference

ACC ACCESS one record for each accession number

SMJ SMJYT one record for each status, molecule, journal, year, type, organelle, division, and db structure information

SUB SUBSEQ one record for each parent or sub-sequence

LOC LOCUS one record for each parent sequence

KEY KEYWORDS one record for each keyword

SPEC SPECIES one record for each taxon

SHRT SHORTL mostly, one record for each element of a short list

LNG LONGL one record for each group of SUBINLNG elements of a long list

EXT EXTRACT (for nucleotide databases only) one record for each exon of each subsequence

TXT TEXT one lrtxt-character record for each label of a species, keyword, or SMJYT

Value

The record count of ACNUC index file, or NA if missing (typically when asking for type = EXT on a protein database).

Author(s)

J.R. Lobry

References

See ACNUC physical structure at http://pbil.univ-lyon1.fr/databases/acnuc/structure.html.

```
citation("seqinr")
```

See Also

choosebank

Examples

```
## Not run:
# Need internet connection
  choosebank("genbank")
  allowedtype <- readfirstrec()
  sapply(allowedtype, function(x) readfirstrec(type = x))

## End(Not run)
```

`readsmj` *Low level function to read ACNUC SMJYT index files*

Description

Extract informations from the SMJYT index file for status, molecule, journal, year, type, organelle, division, and db structure information.

Usage

```
readsmj(socket = autosocket(), num = 2, nl = 10, recnum.add = FALSE, nature.add = T
plong.add = FALSE, libel.add = FALSE, sname.add = FALSE, all.add = FALSE)
```

Arguments

`socket`	an object of class `sockconn` connecting to a remote ACNUC database (default is a socket to the last opened database).
`num`	rank number of first record.
`nl`	number of records to read.
`recnum.add`	to extract record numbers.
`nature.add`	to extract as a factor with human understandable levels the nature of the name. Unordered levels are: status, molecule, journal, year, type, organelle, division and dbstrucinfo.
`plong.add`	to extract the plong.
`libel.add`	to extract the label of the name.
`sname.add`	to extract the short version of the name, that is without the first two characters.
`all.add`	to extract all (all flags set to TRUE).

Value

A data.frame with requested columns.

Author(s)

J.R. Lobry

References

See ACNUC physical structure at: http://pbil.univ-lyon1.fr/databases/acnuc/structure.html.

```
citation("seqinr")
```

See Also

choosebank to start a session and readfirstrec to get the total number of records.

---

rearranged.oriloc *Detection of replication-associated effects on base composition asymmetry in prokaryotic chromosomes.*

Description

Detection of replication-associated effects on base composition asymmetry in prokaryotic chromosomes.

Usage

```
rearranged.oriloc(seq.fasta = system.file("sequences/ct.fasta",package = "seqinr"),
    g2.coord = system.file("sequences/ct.coord",package = "seqinr"))
```

Arguments

seq.fasta	The path of the file containing a FASTA-format sequence. Default value: system.file("sequences/ct.fasta",package = "seqinr") - the FASTA sequence of the Chlamydia trachomatis chromosome.
g2.coord	The path of the file containing the coordinates of the protein coding genes found on this chromosome. This file can be obtained using the function gbk2g2. The format of the file is similar to the output of the Glimmer2 program. The first column contains the index or the name of the gene, the second one contains the start position and the third column contains the end position. For reverse transcribed genes, the start position is greater than the end position.

Details

The purpose of this method is to decouple replication-related and coding sequence-related effects on base composition asymmetry. In order to do so, the analyzed chromosome is artificially rearranged to obtain a perfect gene orientation bias - all forward transcribed genes on the first half of the chromosome, and all reverse transcribed genes on the other half. This rearrangement conserves the relative order of genes within each of the two groups - both forward-encoded and reverse-encoded genes are placed on the rearranged chromosome in increasing order of their coordinates on the real chromosome. If the replication mechanism has a significant effect on base composition asymmetry, this should be seen as a change of slope in the nucleotide skews computed on the rearranged chromosome; the change of slope should take place at the origin or the terminus of replication. Use `extract.breakpoints` to detect the position of the changes in slope on the rearranged nucleotide skews.

Value

A data.frame with six columns: `meancoord.rearr` contains the gene index on the rearranged chromosome; `gcskew.rearr` contains the normalized GC-skew ((G-C)/(G+C)) computed on the third codon positions of protein coding genes, still on the rearranged chromosome; `atskew.rearr` contains the normalized AT-skew ((A-T)/(A+T)) computed on the third codon positions of protein coding genes; `strand.rearr` contains the transcription strand of the gene (either "forward" or "reverse"); `order` contains the permutation that was used to obtain a perfect gene orientation bias; `meancoord.real` contains the mid-coordinate of the genes on the real chromosome (before the rearrangement).

Author(s)

A. Necsulea

References

Necsulea, A. and Lobry, J.R. (2007) A New Method for Assessing the Effect of Replication on DNA Base Composition Asymmetry. *Molecular Biology and Evolution*, **24**:2169-2179.

See Also

`oriloc`, `draw.rearranged.oriloc`, `extract.breakpoints`

Examples

```
### Example for Chlamydia trachomatis ####

### Rearrange the chromosome and compute the nucleotide skews ###

## Not run:
r.ori <- rearranged.oriloc(seq.fasta = system.file("sequences/ct.fasta",package = "seqinr"),
    g2.coord = system.file("sequences/ct.coord",package = "seqinr"))
## End(Not run)

### Extract the breakpoints for the rearranged nucleotide skews ###
```

```
## Not run: breaks <- extract.breakpoints(r.ori, type = c("gcfw", "gcrev"), nbreaks =c(2, 2)

### Draw the rearranged nucleotide skews and place the position of the breakpoints on the gr

## Not run: draw.rearranged.oriloc(r.ori, breaks.gcfw = breaks$gcfw$breaks, breaks.gcrev = b
```

`residuecount` *Total number of residues in an ACNUC list*

Description

Computes the total number of residues (nucleotides or aminoacids) in all sequences of the list of specified rank.

Usage

```
residuecount(lrank, socket = autosocket())
```

Arguments

`lrank`	the list rank on the ACNUC server
`socket`	an object of class `sockconn` connecting to a remote ACNUC database (default is a socket to the last opened database).

Value

A single numeric value corresponding to the total number of residues or NA in case of problem.

Author(s)

J.R. Lobry

References

http://pbil.univ-lyon1.fr/databases/acnuc.html

```
citation("seqinr")
```

See Also

`choosebank`, `query`, `glr`

Examples

```
## Not run:

  ### Need internet connection
  choosebank("emblTP")
  query("mylist", "t=CDS", virtual = TRUE)
  stopifnot(residuecount(glr("mylist")) == 1611439240)
  stopifnot(is.na(residuecount(glr("unknowlist")))) # A warning is issued

## End(Not run)
```

---

revaligntest *Three aligned nucleic acid sequences*

Description

This dataset is used as a sanity check in reverse.align.

Usage

```
data(revaligntest)
```

Format

An object of class alignment with 3 sequences.

References

```
citation("seqinr")
```

See Also

reverse.align

Examples

```
data(revaligntest)
```

`reverse.align` *Reverse alignment - from protein sequence alignment to nucleic sequence alignment*

Description

This function produces an alignment of nucleic protein-coding sequences, using as a guide the alignment of the corresponding protein sequences.

Usage

```
reverse.align(nucl.file, protaln.file, input.format = 'fasta', out.file,
  output.format = 'fasta', align.prot = FALSE, numcode = 1,
  clustal.path = NULL, forceDNAtolower = TRUE, forceAAtolower = FALSE)
```

Arguments

`nucl.file`	A character string specifying the name of the FASTA format file containing the nucleotide sequences.
`protaln.file`	A character string specifying the name of the file containing the aligned protein sequences. This argument must be provided if `align.prot` is set to `FALSE`.
`input.format`	A character string specifying the format of the protein alignment file : 'mase', 'clustal', 'phylip', 'fasta' or 'msf'.
`out.file`	A character string specifying the name of the output file.
`output.format`	A character string specifying the format of the output file. Currently the only implemented format is 'fasta'.
`align.prot`	Boolean. If TRUE, the nucleic sequences are translated and then the protein sequences are aligned with the ClustalW program. The path of the ClustalW binary must also be given (`clustal.path`)
`numcode`	The NCBI genetic code number for the translation of the nucleic sequences. By default the standard genetic code is used.
`clustal.path`	The path of the ClustalW binary. This argument only needs to be setif `align.prot` is TRUE.
`forceDNAtolower`	logical passed to `read.fasta` for reading the nucleic acid file.
`forceAAtolower`	logical passed to `read.alignment` for reading the aligned protein sequence file.

Details

This function an alignment of nucleic protein-coding sequences using as a guide the alignment of the corresponding protein sequences. The file containing the nucleic sequences is given in the compulsory argument 'nucl.file'; this file must be written in the FASTA format.

The alignment of the protein sequences can either be provided directly, trough the 'protaln.file' parameter, or reconstructed with ClustalW, if the parameter 'align.prot' is set to TRUE. In the latter case, the pathway of the ClustalW binary must be given in the 'clustal.path' argument.

The protein and nucleic sequences must have the same name in the files `nucl.file` and `protaln.file`.

The reverse-aligned nucleotide sequences are written to the file specified in the compulsory 'out.file' argument. For now, the only output format implemented is FASTA.

Warning: the 'align.prot=TRUE' option has only been tested on LINUX operating systems. ClustalW must be installed on your system in order for this to work.

Value

NULL

Author(s)

A. Necsulea

References

`citation('seqinr')`

See Also

`read.alignment`, `read.fasta`, `write.fasta`

Examples

```
#
# Read example 'bordetella.fasta': a triplet of orthologous genes from
# three bacterial species (Bordetella pertussis, B. parapertussis and
# B. bronchiseptica):
#

nucl.file <- system.file('sequences/bordetella.fasta', package = 'seqinr')
triplet <- read.fasta(nucl.file)

#
# For this example, 'bordetella.pep.aln' contains the aligned protein
# sequences, in the Clustal format:
#

protaln.file <- system.file('sequences/bordetella.pep.aln', package = 'seqinr')
triplet.pep<- read.alignment(protaln.file, format = 'clustal')

#
# Call reverse.align for this example:
#

reverse.align(nucl.file = nucl.file, protaln.file = protaln.file,
                     input.format = 'clustal', out.file = 'test.revalign')
```

```
#
# Simple sanity check against expected result:
#

res.new <- read.alignment("test.revalign", format = "fasta")
data(revaligntest)
stopifnot(identical(res.new, revaligntest))

#
# Alternatively, we can use ClustalW to align the translated nucleic
# sequences. Here the ClustalW program is accessible simply by the
# 'clustalw' name.
#

## Not run:
reverse.align(nucl.file = nucl.file, out.file = 'test.revalign.clustal',
  align.prot = TRUE, clustal.path = 'clustalw')
## End(Not run)
```

`rot13` *Ergheaf gur EBG-13 pvcurevat bs n fgevat*

Description

rot13 applied to the above title returns the string "Returns the ROT-13 ciphering of a string".

Usage

```
rot13(string)
```

Arguments

`string`	a string of characters.

Value

a string of characters.

Author(s)

J.R. Lobry

References

```
citation("seqinr")
```

See Also

`chartr`

Examples

```
##
## Simple ciphering of a string:
##
message <- "Hello, world!"
rot13(message) # "Uryyb, jbeyq!"
##
## Routine sanity check:
##
stopifnot(identical(rot13(rot13(message)), message))
```

---

s2c *conversion of a string into a vector of chars*

Description

This is a simple utility function to convert a single string such as `"BigBang"` into a vector of chars such as `c("B", "i", "g", "B", "a", "n", "g")`.

Usage

```
s2c(string)
```

Arguments

`string`	a string of chars

Value

a vector of chars. If supplied argument is not a single string, a warning is issued and NA returned.

Author(s)

J.R. Lobry

References

```
citation("seqinr")
```

See Also

`c2s`

Examples

```
stopifnot(all(s2c("BigBang") == c("B", "i", "g", "B", "a", "n", "g")))
```

s2n *simple numerical encoding of a DNA sequence.*

Description

By default, if no `levels` arguments is provided, this function will just code your DNA sequence in integer values following the lexical order `(a > c > g > t)`, that is 0 for "a", 1 for "c", 2 for "g", 3 for "t" and NA for ambiguous bases.

Usage

```
s2n(seq, levels = s2c("acgt"), base4 = TRUE, forceToLower = TRUE)
```

Arguments

`seq`	the sequence as a vector of single chars
`levels`	allowed char values, by default a, c, g and t
`base4`	if TRUE the numerical encoding will start at O, if FALSE at 1
`forceToLower`	if TRUE the sequence is forced to lower case caracters

Value

a vector of integers

Note

The idea of starting numbering at 0 by default is that it enforces a kind of isomorphism between the paste operator on DNA chars and the + operator on integer coding for DNA chars. By this way, you can work either in the char set, either in the integer set, depending on what is more convenient for your purpose, and then switch from one set to the other one as you like.

Author(s)

J.R. Lobry

References

```
citation("seqinr")
```

See Also

`n2s`, `factor`, `unclass`

Examples

```
##
## Example of default behaviour:
##
urndna <- s2c("acgt")
seq <- sample( urndna, 100, replace = TRUE ) ; seq
s2n(seq)
##
## How to deal with RNA:
##
urnrna <- s2c("acgt")
seq <- sample( urnrna, 100, replace = TRUE ) ; seq
s2n(seq)
##
## what happens with unknown characters:
##
urnmess <- c(urndna,"n")
seq <- sample( urnmess, 100, replace = TRUE ) ; seq
s2n(seq)
##
## How to change the encoding for unknown characters:
##
tmp <- s2n(seq) ; tmp[is.na(tmp)] <- -1; tmp
##
## Simple sanity check:
##
stopifnot(all(s2n(s2c("acgt")) == 0:3))
```

---

savelist *Save sequence names or accession numbers into a file*

Description

This function retrieves all sequence names or all accession number from an ACNUC list and saves them into a file.

Usage

```
savelist(lrank, type = c("N", "A"),
                      filename = paste(gln(lrank), ifelse(type == "N", "mne", "acc")
                      socket = autosocket(), warnme = TRUE)
```

Arguments

lrank	the rank of the ACNUC list to consider.
type	use "N" for sequence names (mnemonics) and "A" for accession numbers. Default is "N".
filename	a string of character giving the name of the file to save results.

socket	an object of class sockconn connecting to a remote ACNUC database (default is a socket to the last opened database).
warnme	if TRUE a message is issued on the console when complete.

Value

none.

Author(s)

J.R. Lobry

References

http://pbil.univ-lyon1.fr/databases/acnuc.html

```
citation("seqinr")
```

See Also

choosebank, query, glr to get a list rank from its name, clfcd for the inverse operation of savelist

Examples

```
## Not run:

  ### Need internet connection
  choosebank("emblTP")
  query("mylist", "sp=felis catus et t=cds", virtual=TRUE)
  savelist(glr("mylist"))
  # 603 sequence mnemonics written into file: MYLIST.mne
  savelist(glr("mylist"), type = "A")
  # 603 sequence accession numbers written into file: MYLIST.acc

## End(Not run)
```

---

seqinr-package *Biological Sequences Retrieval and Analysis*

Description

Exploratory data analysis and data visualization for biological sequence (DNA and protein) data. Include also utilities for sequence data management under the ACNUC system.

Author(s)

Delphine Charif and Jean R. Lobry and Anamaria Necsulea and Leonor Palmeira

Maintainer: Simon Penel <penel@biomserv.univ-lyon1.fr>

References

citation('seqinr')

`setlistname` *Sets the name of an ACNUC list identified by its rank*

Description

This is a low level function to set the name of a list from an ACNUC server. It should not be used directly by end users.

Usage

```
setlistname(lrank, name = "list1", socket = autosocket())
```

Arguments

`lrank`	the list rank on the ACNUC server
`name`	the name to use for this list
`socket`	an object of class `sockconn` connecting to a remote ACNUC database (default is a socket to the last opened database).

Value

A single numeric value corresponding to:

`NA`	Empty answer from server.
`0`	OK.
`3`	if another list with that name already existed and was deleted.
`4`	no list of rank `lrank` exists.

Author(s)

J.R. Lobry

References

http://pbil.univ-lyon1.fr/databases/acnuc.html

```
citation("seqinr")
```

See Also

choosebank, query, glr

Examples

```
## Not run:

  ### Need internet connection
  choosebank("emblTP")
  query("mylist", "sp=felis catus et t=CDS", virtual = TRUE)
  # Change list name on server:
  setlistname(lrank = glr("mylist"), name = "feliscatus") # 0, OK.
  glr("mylist") # 0, list doesn't exist no more.
  glr("feliscatus") # 2, this list exists.
  # Note the danger here: the object mylist is still present in the user workspace
  # while the corresponding list was deleted from server.

## End(Not run)
```

---

splitseq *split a sequence into sub-sequences*

Description

Split a sequence into sub-sequences of 3 (the default size) with no overlap between the sub-sequences.

Usage

```
splitseq(seq, frame = 0, word = 3)
```

Arguments

seq	a vector of chars
frame	an integer (0, 1, 2) giving the starting position to split the sequence
word	an integer giving the size of the sub-sequences

Value

This function returns a vector which contains the sub-sequences.

Author(s)

J.R. Lobry

References

```
citation("seqinr")
```

See Also

split

Examples

```
cds <- s2c("aacgttgcaggtcgctcgctacgtagctactgttt")
#
# To obtain the codon sequence in frame 0:
#
stopifnot(identical(splitseq(cds),
  c("aac", "gtt", "gca", "ggt", "cgc", "tcg", "cta", "cgt", "agc", "tac", "tgt")))
#
# Show the effect of frame and word with a ten char sequence:
#
(tenchar <- s2c("1234567890"))
splitseq(tenchar, frame = 0)
splitseq(tenchar, frame = 1)
splitseq(tenchar, frame = 2)
splitseq(tenchar, frame = 0, word = 2)
splitseq(tenchar, frame = 0, word = 1)
```

---

stresc *Utility function to escape LaTeX special characters present in a string*

Description

This function returns a vector of strings in which LaTeX special characters are escaped, this is useful in conjunction with xtable.

Usage

```
stresc(strings)
```

Arguments

strings	A vector of strings to deal with.

normal-bracket8bracket-normal

Value

Returns a vector of strings with escaped characters within each string.

Author(s)

J.R. Lobry

References

```
citation("seqinr")
```

See Also

s2c

Examples

```
stresc("MISC_RNA")
stresc(c("BB_0001","BB_0002"))
```

`syncodons` *Synonymous codons*

Description

Returns all synonymous codons for each codon given

Usage

```
syncodons(codons, numcode = 1)
```

Arguments

`codons`	A sequence of codons as generated by `splitseq`
`numcode`	The genetic code number as in `translate`

Value

a list containing, for each codon given (list tags), all synonymous codons (including the original one)

Author(s)

Leonor Palmeira, J.R. Lobry

References

```
citation("seqinr")
```

See Also

`synsequence`

Examples

```
#
# The four synonymous codons for Alanine in the standard genetic code:
#
syncodons("ggg")
#
# With a sequence:
#
toycds <- s2c("tctgagcaaataaatcgg")
syncodons(splitseq(toycds))
```

```
#
# Sanity check with the standard genetic code:
#
stdgencode <- structure(list(
  ttt = c("ttc", "ttt"),
  ttc = c("ttc", "ttt"),
  tta = c("cta", "ctc", "ctg", "ctt", "tta", "ttg"),
  ttg = c("cta", "ctc", "ctg", "ctt", "tta", "ttg"),
  tct = c("agc", "agt", "tca", "tcc", "tcg", "tct"),
  tcc = c("agc", "agt", "tca", "tcc", "tcg", "tct"),
  tca = c("agc", "agt", "tca", "tcc", "tcg", "tct"),
  tcg = c("agc", "agt", "tca", "tcc", "tcg", "tct"),
  tat = c("tac", "tat"),
  tac = c("tac", "tat"),
  taa = c("taa", "tag", "tga"),
  tag = c("taa", "tag", "tga"),
  tgt = c("tgc", "tgt"),
  tgc = c("tgc", "tgt"),
  tga = c("taa", "tag", "tga"),
  tgg = "tgg",
  ctt = c("cta", "ctc", "ctg", "ctt", "tta", "ttg"),
  ctc = c("cta", "ctc", "ctg", "ctt", "tta", "ttg"),
  cta = c("cta", "ctc", "ctg", "ctt", "tta", "ttg"),
  ctg = c("cta", "ctc", "ctg", "ctt", "tta", "ttg"),
  cct = c("cca", "ccc", "ccg", "cct"),
  ccc = c("cca", "ccc", "ccg", "cct"),
  cca = c("cca", "ccc", "ccg", "cct"),
  ccg = c("cca", "ccc", "ccg", "cct"),
  cat = c("cac", "cat"),
  cac = c("cac", "cat"),
  caa = c("caa", "cag"),
  cag = c("caa", "cag"),
  cgt = c("aga", "agg", "cga", "cgc", "cgg", "cgt"),
  cgc = c("aga", "agg", "cga", "cgc", "cgg", "cgt"),
  cga = c("aga", "agg", "cga", "cgc", "cgg", "cgt"),
  cgg = c("aga", "agg", "cga", "cgc", "cgg", "cgt"),
  att = c("ata", "atc", "att"),
  atc = c("ata", "atc", "att"),
  ata = c("ata", "atc", "att"),
  atg = "atg",
  act = c("aca", "acc", "acg", "act"),
  acc = c("aca", "acc", "acg", "act"),
  aca = c("aca", "acc", "acg", "act"),
  acg = c("aca", "acc",  "acg", "act"),
  aat = c("aac", "aat"),
  aac = c("aac", "aat"),
  aaa = c("aaa", "aag"),
  aag = c("aaa", "aag"),
  agt = c("agc", "agt", "tca", "tcc", "tcg", "tct"),
  agc = c("agc", "agt", "tca", "tcc", "tcg", "tct"),
  aga = c("aga", "agg", "cga", "cgc", "cgg", "cgt"),
  agg = c("aga", "agg", "cga", "cgc", "cgg", "cgt"),
  gtt = c("gta", "gtc", "gtg", "gtt"),
```

```
  gtc = c("gta", "gtc", "gtg", "gtt"),
  gta = c("gta", "gtc", "gtg", "gtt"),
  gtg = c("gta", "gtc", "gtg", "gtt"),
  gct = c("gca", "gcc", "gcg", "gct"),
  gcc = c("gca", "gcc", "gcg", "gct"),
  gca = c("gca", "gcc", "gcg", "gct"),
  gcg = c("gca", "gcc", "gcg", "gct"),
  gat = c("gac", "gat"),
  gac = c("gac", "gat"),
  gaa = c("gaa", "gag"),
  gag = c("gaa", "gag"),
  ggt = c("gga", "ggc", "ggg", "ggt"),
  ggc = c("gga", "ggc", "ggg", "ggt"),
  gga = c("gga", "ggc", "ggg", "ggt"),
  ggg = c("gga", "ggc", "ggg", "ggt")),

.Names = c("ttt", "ttc", "tta", "ttg", "tct", "tcc", "tca", "tcg", "tat", "tac",
"taa", "tag", "tgt", "tgc", "tga", "tgg", "ctt", "ctc", "cta",
"ctg", "cct", "ccc", "cca", "ccg", "cat", "cac", "caa", "cag",
"cgt", "cgc", "cga", "cgg", "att", "atc", "ata", "atg", "act",
"acc", "aca", "acg", "aat", "aac", "aaa", "aag", "agt", "agc",
"aga", "agg", "gtt", "gtc", "gta", "gtg", "gct", "gcc", "gca",
"gcg", "gat", "gac", "gaa", "gag", "ggt", "ggc", "gga", "ggg"))
#
# Now the check:
#
currentresult <- syncodons(words(alphabet = s2c("tcag")))
stopifnot(identical(stdgencode, currentresult))
```

`synsequence` *Random synonymous coding sequence generation*

Description

Generates a random synonymous coding sequence, according to a certain codon usage bias

Usage

```
synsequence(sequence, numcode = 1, ucoweight = NULL)
```

Arguments

`sequence`	A nucleic acids sequence
`numcode`	The genetic code number as in `translate`
`ucoweight`	A list of weights containing the desired codon usage bias as generated by `ucoweight`

Value

a sequence translating to the same protein sequence as the original one (cf. `translate`), but containing synonymous codons

Author(s)

Leonor Palmeira

References

```
citation("seqinr")
```

See Also

ucoweight

Examples

```
data(ec999)
sequence=ec999[1][[1]]
synsequence(sequence,1,ucoweight(sequence))
```

---

tablecode *to plot genetic code as in textbooks*

Description

This function plots a genetic code table as in textbooks, that is following the order `T > C > A > G` so that synonymous codons are almost always in the same boxes.

Usage

```
tablecode(numcode = 1, urn.rna = s2c("TCAG"), dia = FALSE, latexfile = NULL,
label = latexfile, size = "normalsize", caption = NULL,
preaa = rep("", 64), postaa = rep("", 64),
precodon = preaa, postcodon = postaa)
```

Arguments

`numcode`	The genetic code number as in `translate`
`urn.rna`	The letters to display codons, use s2c("UCAG") if you want the code in terms of RNA sequence
`latexfile`	The name of a LaTex file if you want to redirect the output
`label`	The label for the LaTeX table
`size`	The LaTex size of characters for the LaTeX table
`preaa`	A string to insert before the amino-acid in the LaTeX table
`postaa`	A string to insert after the amino-acid in the LaTeX table
`precodon`	A string to insert before the codon in the LaTeX table
`postcodon`	A string to insert after the codon in the LaTeX table
`caption`	The caption of the LaTeX table
`dia`	to produce a yellow/blue plot for slides

Details

The codon order for `preaa`, `postaa`, `precodon`, and `postcodon` should be the same as in `paste(paste(rep(s2c("tcag"), each =16), s2c("tcag"), sep = ""), rep(s2c("tcag"), each = 4), sep = "")`

Author(s)

J.R. Lobry

References

```
citation("seqinr")
```

See Also

`translate`, `syncodons`

Examples

```
#
# Show me the standard genetic code:
#

 tablecode()
```

---

toyaa *A toy example of amino-acid counts in three proteins*

Description

This is a toy data set to illustrate the importance of metric choice.

Usage

```
data(toyaa)
```

Format

A data frame with 3 observations on the following 3 variables:

Ala Alanine counts

Val Valine counts

Cys Cysteine counts

Source

This toy example was inspired by Gautier, C: Analyses statistiques et évolution des séquences d'acides nucléiques. PhD thesis (1987), Université Claude Bernard - Lyon I.

References

```
citation("seqinr")
```

Examples

```
data(toyaa)
```

`toycodon` *A toy example of codon counts in three coding sequences*

Description

This is a toy data set to illustrate synonymous and non-synonymous codon usage analyses.

Usage

```
data(toyaa)
```

Format

A data frame with 3 observations (coding sequences) for 10 codons.

Source

Created for release 1.0-4 of seqinr's vignette.

References

```
citation("seqinr")
```

Examples

```
data(toycodon)
```

`translate` *Translate nucleic acid sequences into proteins*

Description

This function translates nucleic acid sequences into the corresponding peptide sequence. It can translate in any of the 3 forward or three reverse sense frames. In the case of reverse sense, the reverse-complement of the sequence is taken. It can translate using the standard (universal) genetic code and also with non-standard codes. Ambiguous bases can also be handled.

Usage

```
translate(seq, frame = 0, sens = "F", numcode = 1, NAstring = "X", ambiguous = FALS
```

Arguments

`seq`	the sequence to translate as a vector of single characters in lower case letters.
`frame`	Frame(s) (0,1,2) to translate. By default the frame `0` is used.
`sens`	Sense to translate: `F` for forward sense and `R` for reverse sense.
`numcode`	The ncbi genetic code number for translation. By default the standard genetic code is used.
`NAstring`	How to translate amino-acids when there are ambiguous bases in codons.
`ambiguous`	If TRUE, ambiguous bases are taken into account so that for instance GGN is translated to Gly in the standard genetic code.

Details

The following genetic codes are described here. The number preceding each code corresponds to `numcode`.

1 standard
2 vertebrate.mitochondrial
3 yeast.mitochondrial
4 protozoan.mitochondrial+mycoplasma
5 invertebrate.mitochondrial
6 ciliate+dasycladaceal
9 echinoderm+flatworm.mitochondrial
10 euplotid
11 bacterial+plantplastid
12 alternativeyeast
13 ascidian.mitochondrial
14 alternativeflatworm.mitochondrial

15 blepharism

16 chlorophycean.mitochondrial

21 trematode.mitochondrial

22 scenedesmus.mitochondrial

23 hraustochytrium.mitochondria

Value

`translate` returns a vector of single characters containing the peptide sequence in the standard one-letter IUPAC code. Termination (STOP) codons are translated by the character '*'.

Author(s)

D. Charif, J.R. Lobry

References

The genetic codes have been taken from the ncbi taxonomy database: http://www.ncbi.nlm.nih.gov/Taxonomy/Utils/wprintgc.cgi?mode=c. Last update October 05, 2000.
The IUPAC one-letter code for aminoacids is described at: http://www.chem.qmul.ac.uk/iupac/AminoAcid/

```
citation("seqinr")
```

See Also

Use `tolower` to change upper case letters into lower case letters. For coding sequences obtained from an ACNUC server with `query` it's better to use the function `getTrans` so that the relevant genetic code and the relevant frame are automatically used. The genetic codes are given in the object `SEQINR.UTIL`, a more human readable form is given by the function `tablecode`. Use `aaa` to get the three-letter code for amino-acids.

Examples

```
##
## Toy CDS example invented by Leonor Palmeira:
##
toycds <- s2c("tctgagcaaataaatcgg")
translate(seq = toycds) # should be c("S", "E", "Q", "I", "N", "R")
##
## Toy CDS example with ambiguous bases:
##
toycds2 <- s2c("tcngarcarathaaycgn")
translate(toycds2) # should be c("X", "X", "X", "X", "X", "X")
translate(toycds2, ambiguous = TRUE) # should be c("S", "E", "Q", "I", "N", "R")
translate(toycds2, ambiguous = TRUE, numcode = 2) # should be c("S", "E", "Q", "X", "N", "R"
##
## Real CDS example:
##
realcds <- read.fasta(file = system.file("sequences/malM.fasta", package ="seqinr"))[[1]]
```

```
translate(seq = realcds)
# Biologically correct, only one stop codon at the end
translate(seq = realcds, frame = 3, sens = "R", numcode = 6)
# Biologically meaningless, note the in-frame stop codons

## Not run:
## Need internet connection.
## Translation of the following EMBL entry:
##
## FT   CDS             join(complement(153944..154157),complement(153727..153866),
## FT                   complement(152185..153037),138523..138735,138795..138955)
## FT                   /codon_start=1
## FT                   /db_xref="FLYBASE:FBgn0002781"
## FT                   /db_xref="GOA:Q86B86"
## FT                   /db_xref="TrEMBL:Q86B86"
## FT                   /note="mod(mdg4) gene product from transcript CG32491-RZ;
## FT                   trans splicing"
## FT                   /gene="mod(mdg4)"
## FT                   /product="CG32491-PZ"
## FT                   /locus_tag="CG32491"
## FT                   /protein_id="AAO41581.1"
## FT                   /translation="MADDEQFSLCWNNFNTNLSAGFHESLCRGDLVDVSLAAEGQIVKA
## FT                   HRLVLSVCSPFFRKMFTQMPSNTHAIVFLNNVSHSALKDLIQFMYCGEVNVKQDALPAF
## FT                   ISTAESLQIKGLTDNDPAPQPPQESSPPPAAPHVQQQQIPAQRVQRQQPRASARYKIET
## FT                   VDDGLGDEKQSTTQIVIQTTAAPQATIVQQQQPQQAAQQIQSQQLQTGTTTTATLVSTN
## FT                   KRSAQRSSLTPASSSAGVKRSKTSTSANVMDPLDSTTETGATTTAQLVPQQITVQTSVV
## FT                   SAAEAKLHQQSPQQVRQEEAEYIDLPMELPTKSEPDYSEDHGDAAGDAEGTYVEDDTYG
## FT                   DMRYDDSYFTENEDAGNQTAANTSGGGVTATTSKAVVKQQSQNYSESSFVDTSGDQGNT
## FT                   EAQVTQHVRNCGPQMFLISRKGGTLLTINNFVYRSNLKFFGKSNNILYWECVQNRSVKC
## FT                   RSRLKTIGDDLYVTNDVHNHMGDNKRIEAAKAAGMLIHKKLSSLTAADKIQGSWKMDTE
## FT                   GNPDHLPKM"
choosebank("emblTP")
query("trans", "N=AE003734.PE35")
getTrans(trans$req[[1]])
## Complex transsplicing operations, the correct frame and the correct
## genetic code are automatically used for translation into protein.
## End(Not run)
```

`trimSpace` *Trim leading and/or trailing spaces in strings*

Description

This function removes from a character vector the longest successive run of space characters starting at the begining of the strings (leading space), or the longest successive run of space characters at the end of the strings (trailing space), or both (and this is the default behaviour).

Usage

```
trimSpace(x, leading = TRUE, trailing = TRUE, space = "[:space:]")
```

Arguments

x	a character vector
leading	logical defaulting to TRUE: should leading spaces be trimed off?
trailing	logical defaulting to TRUE: should trailing spaces be trimed off?
space	an extended regular expression defining space characters

Details

The default value for the space character definition is large: in addition to the usual space, other character such as the tabulation and newline character are considered as space characters. See extended regular expression for a complete list.

Value

a character vector with the same length as x.

Author(s)

J.R. Lobry

References

citation("seqinr").

See Also

Extended regular expressionsare described in regular expression (aka regexp).

Examples

```
  #
  # Simple use:
  #
stopifnot( trimSpace("   seqinR   ") == "seqinR" )

  #
  # Basic use, remove space at both ends:
  #
testspace <- c("   with leading space", "with trailing space   ", "   with both   ")
stopifnot(all( trimSpace(testspace) == c("with leading space", "with trailing space", "with

  #
  # Remove only leading space:
  #
stopifnot(all( trimSpace(testspace, trailing = FALSE) == c("with leading space", "with trail

  #
  # Remove only trailing space:
  #
stopifnot(all( trimSpace(testspace, leading = FALSE) == c("   with leading space", "with tra
```

```
  #
  # This should do nothing:
  #
stopifnot(all( trimSpace(testspace, leading = FALSE, trailing = FALSE) == testspace))

  #
  # How to use alternative space characters:
  #
allspaces <- "\t\n\v\f\r seqinR \t\n\v\f\r"
stopifnot(trimSpace(allspaces) == "seqinR")
stopifnot(trimSpace(allspaces, space = "\t\n") == "\v\f\r seqinR \t\n\v\f\r")
```

`uco` *Codon usage indices*

Description

`uco` calculates some codon usage indices: the codon counts `eff`, the relative frequencies `freq` or the Relative Synonymous Codon Usage `rscu`.

Usage

```
uco(seq, frame = 0, index = c("eff", "freq", "rscu"), as.data.frame = FALSE,
NA.rscu = NA)
```

Arguments

`seq`	a coding sequence as a vector of chars
`frame`	an integer (0, 1, 2) giving the frame of the coding sequence
`index`	codon usage index choice, partial matching is allowed. `eff` for codon counts, `freq` for codon relative frequencies, and `rscu` the RSCU index
`as.data.frame`	logical. If `TRUE`: all indices are returned into a data frame.
`NA.rscu`	when an amino-acid is missing, RSCU are no more defined and repported as missing values (`NA`). You can force them to another value (typically 0 or 1) with this argument.

Details

Codons with ambiguous bases are ignored.

RSCU is a simple measure of non-uniform usage of synonymous codons in a coding sequence (Sharp *et al.* 1986). RSCU values are the number of times a particular codon is observed, relative to the number of times that the codon would be observed for a uniform synonymous codon usage (i.e. all the codons for a given amino-acid have the same probability). In the absence of any codon usage bias, the RSCU values would be 1.00 (this is the case for sequence `cds` in the exemple thereafter).

A codon that is used less frequently than expected will have an RSCU value of less than 1.00 and vice versa for a codon that is used more frequently than expected.

Do not use correspondence analysis on RSCU tables as this is a source of artifacts (Perriere and Thioulouse 2002). Within-aminoacid correspondence analysis is a simple way to study synonymous codon usage (Charif *et al.* 2005).

If `as.data.frame` is FALSE, `uco` returns one of these:

eff a table of codon counts

freq a table of codon relative frequencies

rscu a numeric vector of relative synonymous codon usage values

If `as.data.frame` is TRUE, `uco` returns a data frame with five columns:

aa a vector containing the name of amino-acid

codon a vector containing the corresponding codon

eff a numeric vector of codon counts

freq a numeric vector of codon relative frequencies

rscu a numeric vector of RSCU index

Value

If `as.data.frame` is FALSE, the default, a table for `eff` and `freq` and a numeric vector for `rscu`. If `as.data.frame` is TRUE, a data frame with all indices is returned.

Author(s)

D. Charif, J.R. Lobry, G. Perriere

References

```
citation("seqinr")
```

Sharp, P.M., Tuohy, T.M.F., Mosurski, K.R. (1986) Codon usage in yeast: cluster analysis clearly differentiates highly and lowly expressed genes. *Nucl. Acids. Res.*, **14**:5125-5143.

Perriere, G., Thioulouse, J. (2002) Use and misuse of correspondence analysis in codon usage studies. *Nucl. Acids. Res.*, **30**:4548-4555.

Charif, D., Thioulouse, J., Lobry, J.R., Perriere, G. (2005) Online Synonymous Codon Usage Analyses with the ade4 and seqinR packages. *Bioinformatics*, **21**:545-547. http://pbil.univ-lyon1.fr/members/lobry/repro/bioinfo04/.

Examples

```
## Show all possible codons:
words()

## Make a coding sequence from this:
(cds <- s2c(paste(words(), collapse = "")))

## Get codon counts:
uco(cds, index = "eff")

## Get codon relative frequencies:
uco(cds, index = "freq")

## Get RSCU values:
uco(cds, index = "rscu")

## Show what happens with ambiguous bases:
uco(s2c("aaannnttt"))

## Use a real coding sequence:
rcds <- read.fasta(file = system.file("sequences/malM.fasta", package = "seqinr"))[[1]]
uco( rcds, index = "freq")
uco( rcds, index = "eff")
uco( rcds, index = "rscu")
uco( rcds, as.data.frame = TRUE)

## Show what happens with RSCU when an amino-acid is missing:
ecolicgpe5 <- read.fasta(file = system.file("sequences/ecolicgpe5.fasta",package="seqinr"))[
uco(ecolicgpe5, index = "rscu")

## Force NA to zero:
uco(ecolicgpe5, index = "rscu", NA.rscu = 0)
```

---

ucoweight *Weight of each synonymous codon*

Description

Returns a list containing, for each of the 20 amino acids + STOP codon, the codon usage bias of each of the synonymous codon according to a given codon sequence.

Usage

```
ucoweight(sequence, numcode = 1)
```

Arguments

sequence	A nucleic acids sequence
numcode	The genetic code number as in `translate`

Value

a list containing, for each of the 20 amino acids and STOP codon (list tags), the weight of each synonymous codon (including the original one).

Author(s)

Leonor Palmeira

References

```
citation("seqinr")
```

See Also

synsequence

Examples

```
data(ec999)
ucoweight(ec999[1][[1]])
```

---

waterabs *Light absorption by the water column*

Description

The absorption of light by water is highly dependent on the wavelength, this dataset gives the absorption coefficients from 200 to 700 nm.

Usage

```
data(waterabs)
```

Format

A data.frame with 2 columns:

lambda wavelength in nm

abs absorption coefficient in 1/cm

Source

Data were compiled by Palmeira (2007) from the cited references.

References

Palmeira, L. (2007) *Analyse et modélisation des dépendances entre sites voisins dans l'évolution des séquences d'ADN*, PhD thesis, Université Claude Bernard - Lyon I.

Litjens R. A., Quickenden T. I. and Freeman C. G. (1999). Visible and near-ultraviolet absorption spectrum of liquid water. *Applied Optics*, **38**:1216-1223.

Quickenden T. I. & Irvin J. A. (1980). The ultraviolet absorption spectrum of liquid water. *The Journal of Chemical Physics*, **72**:4416-4428.

```
citation("seqinr")
```

Examples

```
data(waterabs)

d <- 100*seq(from = 0, to = 150, by = 1) # depth in cm
lambda <- waterabs$lambda                # wavelength in nm
abs <- waterabs$absorption               # absorption coefficient cm-1
#
# Smooth signal with cubic splines
#
  tmp <- spline(lambda, abs, n = 255)
  lambda <- tmp$x
  abs <- tmp$y

  zun <- sapply(abs,function(x) 10^(-x*d))
  z <- sapply(nrow(zun):1, function(x) zun[x,])
#
# Set up world coordinates:
#
  plot.new()
  plot.window(xlim = range(lambda), ylim = range(d), xaxs = "i",  yaxs = "i")
#
# Annotate:
#
  title(ylab = 'Depth under water surface (m)', xlab = "Wavelength (nm)",
  main = "Light absorption by the water column")
  axis(2 , at = seq(0, 15000, l = 7),
      labels = rev(c("0","25","50","75","100","125","150")), las = 1)
  axis(1,at=(3:6)*100,labels= TRUE)
#
# Show me rainbow colors:
#
  alpha <- 1
  coul=c(rep(rgb(1,1,1, alpha = alpha), 181),
    rev(hsv(h=seq(0,5/6,l=320),alpha = alpha)))
  rect(seq(200,699), 0, seq(201,700), 15000 , col = coul, border = coul)
#
# Grey scale:
```

```
#
  ngris <- 5
  image(x = lambda, y = d, z = z, col = rgb(1:ngris, 1:ngris, 1:ngris, alpha = 0.7*(ngris:1)
  max = ngris),
  axes = F, add = TRUE,
  breaks = seq(from = min(z), to = max(z), length = ngris + 1))

#
# Contour lines:
#
  contour(x = lambda, y = d, z = z, add = TRUE, drawlabels = TRUE,labcex= 0.75,
  col='black',
  levels = seq(from = min(z), to = max(z), length = ngris + 1))
  box()
```

`words` *To get all words from an alphabet.*

Description

Generates a vectors of all the words from a given alphabet, with right positions varying faster, for instance if the `alphabet` is `(c("0","1")` and the `length` is 2 you will obtain `c("00", "01", "10", "11")`

Usage

```
words(length = 3, alphabet = s2c("acgt"))
```

Arguments

`length`	the number of characters in the words
`alphabet`	a vector of characters

Value

A vector of string whith `length` characters.

Author(s)

J.R. Lobry

References

```
citation("seqinr")
```

See Also

`kronecker`, `outer`

Examples

```
#
# Get all 64 codons:
#
stopifnot(all(words() ==
c("aaa", "aac", "aag", "aat", "aca", "acc", "acg", "act", "aga", "agc", "agg", "agt", "ata",
   "caa", "cac", "cag", "cat", "cca", "ccc", "ccg", "cct", "cga", "cgc", "cgg", "cgt", "cta"
   "gaa", "gac", "gag", "gat", "gca", "gcc", "gcg", "gct", "gga", "ggc", "ggg", "ggt", "gta"
   "taa", "tac", "tag", "tat", "tca", "tcc", "tcg", "tct", "tga", "tgc", "tgg", "tgt", "tta"
#
# Get all codons with u c a g for bases:
#
words(alphabet = s2c("ucag"))
#
# Get all tetranucleotides:
#
words(length = 4)
#
# Get all dipeptides:
#
words(length = 2, alphabet = a()[-1])
```

---

words.pos *Positions of possibly degenerated motifs within sequences*

Description

`word.pos` searches all the occurences of the motif `pattern` within the sequence `text` and returns their positions. This function is based on `regexp` allowing thus for complex motif searches. The main difference with `gregexpr` is that non disjoint matches are reported here.

Usage

```
words.pos(pattern, text, ignore.case = FALSE, extended = FALSE,
                      perl = TRUE, fixed = FALSE, useBytes = TRUE, ...)
```

Arguments

`pattern`	character string containing a regular expression (or character string for `fixed = TRUE`) to be matched in the given character vector.
`text`	a character vector where matches are sought.
`ignore.case`	if `FALSE`, the pattern matching is case sensitive and if `TRUE`, case is ignored during matching.
`extended`	if `TRUE`, extended regular expression matching is used, and if `FALSE` basic regular expressions are used.
`perl`	logical. Should perl-compatible regexps be used if available? Has priority over `extended`.

fixed	logical. If TRUE, pattern is a string to be matched as is. Overrides all conflicting arguments.
useBytes	logical. If TRUE the matching is done byte-by-byte rather than character-by-character.
...	arguments passed to regexpr.

Details

Default parameter values have been tuned for speed when working biological sequences.

Value

a vector of positions for which the motif pattern was found in the sequence text.

Author(s)

J.R. Lobry

References

```
citation("seqinr")
```

See Also

regexpr

Examples

```
myseq <- "tatagaga"
words.pos("t", myseq)   # Should be 1 3
words.pos("tag", myseq) # Should be 3
words.pos("ga", myseq)  # Should be 5 7
# How to specify ambiguous base ? Look for YpR motifs by
words.pos("[ct][ag]", myseq) # Should be 1 3
#
# Show the difference with gregexpr:
#
words.pos("toto", "totototo")           # 1 3 5 (three overlapping matches)
unlist(gregexpr("toto",  "totototo")) # 1 5    (two disjoint matches)
```

---

write.fasta *Write file in fasta format*

Description

Writes sequences to a file in FASTA format.

Usage

```
write.fasta(sequences, names, nbchar = 60, file.out, open = "w")
```

Arguments

sequences	A DNA or protein sequence (i.e. a character vector) or a list of sequences (i.e. a list of character vectors).
names	The name(s) of the sequences.
nbchar	The number of characters per line (default: 60)
file.out	The name of the output file.
open	Mode to open the output file, use "w" to write into a new file, use "a" to append at the end of an already existing file.

Value

NULL

Author(s)

A. Necsulea

References

```
citation("seqinr")
```

See Also

read.fasta

Examples

```
## Read sequences from a FASTA file:
ortho <- read.fasta(file = system.file("sequences/ortho.fasta", package =
"seqinr"))

## Select only third codon positions:
ortho3 <- lapply(ortho, function(x) x[seq(from = 3, to = length(x), by = 3)])

## Write the modified sequences to a file:
write.fasta(sequences = ortho3, names = names(ortho3), nbchar = 80, file.out = "ortho3.fasta

## Read it again from the same file and check that sequences are preserved:
ortho3bis <- read.fasta("ortho3.fasta", set.attributes = FALSE)
stopifnot(identical(ortho3bis, ortho3))
```

`dinucleotides` *Statistical over- and under- representation of dinucleotides in a sequence*

Description

These two functions compute two different types of statistics for the measure of statistical dinculeotide over- and under-representation : the rho statistic, and the z-score, each computed for all 16 dinucleotides.

Usage

```
rho(sequence, alphabet = s2c("acgt"))
zscore(sequence, simulations = NULL, modele, exact = FALSE, alphabet = s2c("acgt"),
```

Arguments

`sequence`	A nucleic acids sequence
`simulations`	If `NULL`, analytical solution is computed when available (models `base` and codon). Otherwise, it should be the number of permutations for the z-score computation
`modele`	A string of characters describing the model chosen for the random generation
`exact`	Whether exact analytical calculation or an approximation should be used
`alphabet`	A vector of single characters.
`...`	Optional parameters for specific model permutations are passed on to `permutation` function.

Details

The `rho` statistic, as presented in Karlin S., Cardon LR. (1994), can be computed on each of the 16 dinucleotides. It is the frequence of dinucleotide *xy* divided by the product of frequencies of nucleotide *x* and nucleotide *y*. It is equal to 1.00 when dinucleotide *xy* is formed by pure chance, and it is superior (respectively inferior) to 1.00 when dinucleotide *xy* is over- (respectively under-) represented.

The `zscore` statistic, as presented in Palmeira, L., Guéguen, L. and Lobry JR. (2006). The statistic is the normalization of the `rho` statistic by its expectation and variance according to a given random sequence generation model, and follows the standard normal distribution. This statistic can be computed with several models (cf. `permutation` for the description of each of the models). We provide analytical calculus for two of them: the `base` permutations model and the `codon` permutations model.

The `base` model allows for random sequence generation by shuffling (with/without replacement) of all bases in the sequence. Analytical computations are available for this model: either as an approximation for large sequences (cf. Palmeira, L., Guéguen, L. and Lobry JR. (2006)), either as the exact analytical formulae (cf. Schbath, S. (1995)).

The `position` model allows for random sequence generation by shuffling (with/without replacement) of bases within their position in the codon (bases in position I, II or III stay in position I, II or III in the new sequence.

The `codon` model allows for random sequence generation by shuffling (with/without replacement) of codons. Analytical computation is available for this model (Gautier, C., Gouy, M. and Louail, S. (1985)).

The `syncodon` model allows for random sequence generation by shuffling (with/without replacement) of synonymous codons.

Value

a table containing the computed statistic for each dinucleotide

Author(s)

Leonor Palmeira

References

Gautier, C., Gouy, M. and Louail, S. (1985) Non-parametric statistics for nucleic acid sequence study. *Biochimie*, **67**:449-453.

Karlin S. and Cardon LR. (1994) Computational DNA sequence analysis. *Annu Rev Microbiol*, **48**:619-654.

Schbath, S. (1995) Étude asymptotique du nombre d'occurrences d'un mot dans une chaîne de Markov et application à la recherche de mots de fréquence exceptionnelle dans les séquences d'ADN. *Thèse de l'Université René Descartes, Paris V*

Palmeira, L., Guéguen, L. and Lobry, J.R. (2006) UV-targeted dinucleotides are not depleted in light-exposed Prokaryotic genomes. *Molecular Biology and Evolution*, **23**:2214-2219. http://mbe.oxfordjournals.org/cgi/reprint/23/11/2214

```
citation("seqinr")
```

See Also

`permutation`

Examples

```
sequence <- sample(x = s2c("acgt"), size = 6000, replace = TRUE)
rho(sequence)
zscore(sequence, modele = "base")
zscore(sequence, modele = "base", exact = TRUE)
zscore(sequence, modele = "codon")
zscore(sequence, simulations = 1000, modele = "syncodon")
```

Index