Package ‘ptm’

August 7, 2022

Type Package

Title Analyses of Protein Post-Translational Modifications

Version 0.2.6

Maintainer Juan Carlos Aledo <caledo@uma.es>

Description Contains utilities for the analysis of post-translational modifications (PTMs) in proteins, with particular emphasis on the sulfoxidation of methionine residues. Features include the ability to download,
filter and analyze data from the sulfoxidation database 'MetOSite', and integrate data from other main PTMs
(other databases). Utilities to search and characterize S-aromatic motifs in proteins are also provided.
In addition, functions to analyze sequence environments around modifiable residues in proteins can be found.
For instance, 'ptm' allows to search for amino acids either overrepresented or avoided around the modifiable
residues from the proteins of interest. Functions tailored to test statistical hypothesis related to these differential sequence environments are also implemented. A number of utilities to assess the effect
of the modification/mutation of a given residue on the protein stability, have also been included in this package.
Further and detailed information regarding the methods in this package can be found in (Aledo (2020) <https://metositeptm.com>).

License GPL (>= 2)

URL https://bitbucket.org/jcaledo/ptm, https://metositeptm.com

Encoding UTF-8

LazyData true

RoxygenNote 7.2.1

Depends R (>= 4.0.0)

Imports bio3d (>= 2.3-4), Biostrings, curl, graphics, httr (>= 1.3.1),
igraph, jsonlite, muscle, RCurl, seqinr, stats, utils, xml2

Suggests KEGGREST, knitr, markdown, testthat

NeedsCompilation no

Author Juan Carlos Aledo [aut, cre]

Repository CRAN

Date/Publication 2022-08-07 15:20:17 UTC

R topics documented:

`.get.exepath` *Find Full Paths to Executables*

Description

Finds the path to an executable.

Usage

```
.get.exepath(prg)
```

Arguments

`prg`	name of the executable.

Value

Returns the absolute path.

`.get.url` *Get Web Resource*

Description

Gets a web resource.

Usage

```
.get.url(url, n_tries = 3)
```

Arguments

`url`	url to be reached.
`n_tries,`	number of tries.

Value

Returns the response or an error message.

`aa.at` *Residue Found at the Requested Position*

Description

Returns the residue found at the requested position.

Usage

```
aa.at(at, target, uniprot = TRUE)
```

Arguments

`at`	the position in the primary structure of the protein.
`target`	a character string specifying the UniProt ID of the protein of interest or, alternatively, the sequence of that protein.
`uniprot`	logical, if TRUE the argument 'target' should be an ID.

Details

Please, note that when uniprot is set to FALSE, target can be the string returned by a suitable function, such as get.seq or other.

Value

Returns a single character representing the residue found at the indicated position in the indicated protein.

Author(s)

Juan Carlos Aledo

See Also

is.at(), renum.pdb(), renum.meto(), renum(), aa.comp()

Examples

```
## Not run: aa.at(28, 'P01009')
```

---

aa.comp *Amino Acid Composition*

Description

Returns a table with the amino acid composition of the target protein.

Usage

```
aa.comp(target, uniprot = TRUE, reference = 'human', init = FALSE)
```

Arguments

`target`	a character string specifying the UniProt ID of the protein of interest or, alternatively, the sequence of that protein.
`uniprot`	logical, if TRUE the argument 'target' should be an ID.
`reference`	amino acid frequencies (in percent) of the proteinogenic amino acids to be used as reference. It should be either 'human', 'up' (composition of proteins in UniProt in 2019). Alternatively, the user can pass as argument any vector with 20 values to be used as reference.
`init`	logical, whether remove or not the first residue (initiation methionine) from the sequence.

Value

Returns a list where the first element is a dataframe with the observed and expected frequencies for each amino acid, the second element is the result of the Chi-squared test. In addition, a plot to reflect potential deviations from the reference standard composition is shown.

Author(s)

Juan Carlos Aledo

See Also

is.at(), renum.pdb(), renum.meto(), renum(), aa.at()

Examples

```
aa.comp('MPSSVSWGILLLAGLCCLVPVSLAEDPQGDAAQK', uniprot = FALSE)
```

`abundance` *Protein Abundance Data*

Description

Returns data regarding the abundance of a given protein.

Usage

```
abundance(id, ...)
```

Arguments

`id`	the UniProt identifier of the protein of interest.
`...`	either 'jarkat' or 'hela' if required.

Details

For human proteins, in addition to the abundance in the whole organism (by default), the abundance found in Jurkat or HeLa cells can be requested. The data are obtained from the PaxDb.

Value

A numeric value for the abundance, expressed a parts per million (ppm), of the requested protein.

Author(s)

Juan Carlos Aledo

References

Wang et al. Proteomics 2015, 10.1002/pmic.201400441. (PMID: 25656970)

Examples

```
abundance(id = 'A0AVT1')
## Not run: abundance(id = 'A0AVT1', 'jurkat')
## Not run: abundance(id = 'A0AVT1', 'hela')
```

---

ac.scan — *Scan a Protein in Search of Acetylation Sites*

Description

Scans the indicated protein in search of acetylation sites.

Usage

```
ac.scan(up_id, db = 'all')
```

Arguments

up_id	a character string corresponding to the UniProt ID.
db	the database where to search. It should be one among 'PSP', 'dbPTM', 'all'.

Details

If db = 'all' has been selected, it may happen that the same residue appears in several rows if it is present in different databases.

Value

Returns a dataframe where each row corresponds to an acetylable residue.

Author(s)

Juan Carlos Aledo

References

Hornbeck et al. Nucleic Acids Res. 2019 47:D433-D441, (PMID: 30445427).

Huang et al. Nucleic Acids Res. 2019 47:D298-D308, (PMID: 30418626).

See Also

meto.scan(), p.scan(), me.scan(), ub.scan(), su.scan(), gl.scan(), sni.scan(), ni.scan(), ptm.scan(), reg.scan(), dis.scan()

Examples

```
## Not run: ac.scan('P01009', db = 'PSP')
```

`acc.dssp` *Compute Residue Accessibility and SASA*

Description

Computes the accessibility as well as the SASA for each reside from the indicated protein.

Usage

```
acc.dssp(pdb, aa = 'all')
```

Arguments

`pdb`	is either a PDB id, or the path to a pdb file.
`aa`	one letter code for the amino acid of interest, or 'all' for all the protein residues.

Details

You must have installed DSSP on your system and in the search path for executables.

Value

A dataframe where each row is an individual residue of the selected protein. The variables computed, among others, are: (i) the secondary structure (ss) element to which the residue belongs, (ii) the solvent accessible surface area (sasa) of each residue in square angstrom ($Å^2$), and (iii) the accessibility (acc) computed as the percent of the sasa that the residue X would have in the tripeptide GXG with the polypeptide skeleton in an extended conformation and the side chain in the conformation most frequently observed in proteins.

Author(s)

Juan Carlos Aledo

References

Miller et al (1987) J. Mol. Biol. 196: 641-656 (PMID: 3681970).

Touw et al (2015) Nucl. Ac. Res. 43(Database issue): D364-D368 (PMID: 25352545).

See Also

atom.dpx(), res.dpx(), str.part()

Examples

```
## Not run: acc.dssp('3cwm')
```

`atom.dpx` *Atom Depth Analysis*

Description

Computes the depth from the surface for each protein's atom.

Usage

```
atom.dpx(pdb)
```

Arguments

`pdb`	is either a PDB id, or the path to a pdb file.

Details

This function computes the depth, defined as the distance in angstroms between the target atom and the closest atom on the protein surface. When the protein is composed of several subunits, the calculations are made for both, the atom being part of the complex, and the atom being only part of the polypeptide chain to which it belongs.

Value

A dataframe with the computed depths.

Author(s)

Juan Carlos Aledo

References

Pintar et al. 2003. Bioinformatics 19:313-314 (PMID: 12538266)

See Also

res.dpx(), acc.dssp(), str.part()

Examples

```
## Not run: atom.dpx('1cll')
```

bg.go — *Search GO Terms for Background Set*

Description

Searches the GO terms of the protein contained in a given set.

Usage

```
bg.go(ids)
```

Arguments

`ids`	either a vector containing the UniProt IDs of the background set or the path to the txt file containing the list of IDs acting as background.

Value

Returns a dataframe with two columns (Uniprot ID, GO terms) and as many rows as different proteins there are in the input set.

Author(s)

Juan Carlos Aledo

References

Rhee et al. (2008) Nature Reviews Genetics 9:509–515.

See Also

search.go(), term.go(), get.go(), go.enrich(), gorilla(), net.go()

Examples

```
## Not run: bg.go(c('P01009', 'P01374', 'Q86UP4'))
```

`ddG.profile` *Contribution of a given position to changes in stability*

Description

Represents the sensitivity of a given position to changes in stability of a protein (DDG).

Usage

```
ddG.profile(prot, ch, pos, pH = 7, Te = 25)
```

Arguments

`prot`	either the 4-letter identifier of a PDB structure, or the amino acid sequence (one letter amino acid code) of a protein.
`ch`	a letter identifying the chain of interest.
`pos`	the position, in the primary structure, of the residue to be mutated.
`pH`	a numeric value between 0 and 14.
`Te`	a numeric value indicating the temperature in degrees Celsius.

Details

It must be remembered that DDG > 0 implies destabilizing change and DDG > 0 implies a stabilizing change.

Value

The function returns a dataframe with the DDG values (kcal/mol) for each alternative amino acid, and a barplot grouping the amino acids according to their physicochemical nature.

Author(s)

Juan Carlos Aledo

See Also

foldx.mut(), imutant()

Examples

```
## Not run: ddG.profile(prot = '1pga', ch = 'A', pos = 27)
```

ddG.ptm *PDB Model and Change in Stability of a Modified Protein*

Description

Builds a PDB model of the modified protein and computes the corresponding change in stability.

Usage

```
ddG.ptm(pdb, ch, pos, ptm, dir = 'f', pH = 7)
```

Arguments

pdb	the 4-letter identifier of a PDB structure or the path to a PDB file.
ch	a letter identifying the chain of interest.
pos	the position, in the primary structure, of the residue to be modified.
ptm	the post-translational modification to be considered. It should be one among: 'pSer', 'pThr', 'pTyr', 'MetO-Q', 'MetO-T'.
dir	indicates the direction of the PTM reaction: either forward ('f'), or backward ('b').
pH	a numeric value between 0 and 14.

Details

The current function uses FoldX to build the model of the modified protein. Currently, FoldX does not allow to change Met by MetO, so we use glutamine (Q) or threonine (T) to mimic MetO.

Value

The function computes and returns the DDG (kcal/mol) for the requested modification, defined as DDG = DGmodified - DGunmodified, where DG is the Gibbs free energy for the folding of the protein from its unfolded state. Thus, a positive value means a destabilizing effect, and vice versa. A PDB model containing the modified target is saved in the current directory.

Author(s)

Juan Carlos Aledo

References

Schymkowitz et al (2005) Nucl. Ac. Res. 33:W382-W388.

See Also

imutant(), foldx.mut(), ddG.profile()

Examples

```
## Not run: ddG.ptm('./1u8f_Repair.pdb', 'O', pos = 246, ptm = 'pThr')
```

---

dis.scan *Scan a Protein in Search of Disease-Related PTM Sites*

Description

Scans the indicated protein in search of disease-related PTM sites.

Usage

```
dis.scan(up_id)
```

Arguments

up_id	a character string corresponding to the UniProt ID.

Value

Returns a dataframe where each row corresponds to a residue, and the columns inform about the disease-related modifications.

Author(s)

Juan Carlos Aledo

References

Hornbeck et al. Nucleic Acids Res. 2019 47:D433-D441, (PMID: 30445427).

See Also

meto.scan(), ac.scan(), me.scan(), ub.scan(), su.scan(), gl.scan(), sni.scan(), ni.scan(), ptm.scan(), reg.scan(), p.scan()

Examples

```
## Not run: dis.scan('P31749')
```

`dpx` *Atom Depth Analysis*

Description

Computes the depth from the surface for each protein's atom.

Usage

```
dpx(pdb)
```

Arguments

pdb	is either a PDB id, or the path to a pdb file.

Details

This function computes the depth, defined as the distance in angstroms between the target atom and the closest atom on the protein surface.

Value

A dataframe with the computed depths.

Author(s)

Juan Carlos Aledo

References

Pintar et al. 2003. Bioinformatics 19:313-314 (PMID: 12538266)

See Also

compute.dssp(), atom.dpx(), res.dpx(), acc.dssp(), str.part()

Examples

```
## Not run: dpx('3cwm')
```

env.extract — *Sequence Environment Around a Given Position*

Description

Extracts the sequence environment around a given position.

Usage

```
env.extract(prot, db = 'none', c, r, ctr = 'none', exclude = c())
```

Arguments

prot	either a uniprot id or a string sequence.
db	a character string specifying the desired database; it must be one of 'uniprot', 'metosite', 'none'.
c	center of the environment.
r	radius of the environment.
ctr	the type of control environment; it must be one of 'random', 'closest', or 'none'.
exclude	a vector containing the positions to be excluded as control.

Details

The random control returns an environment center at a random position containing the same type or amino acid than the positive environment. The closest control searches for the closest position where such a type of amino acid is found and returns its environment.

Value

Returns a list of two strings (environments).

Author(s)

Juan Carlos Aledo

References

Aledo et al. Sci Rep. 2015; 5: 16955. (PMID: 26597773)

See Also

env.matrices(), env.Ztest() and env.plot()

Examples

```
env.extract('P01009', db = 'uniprot', 271, 10, ctr = 'random')
```

`env.matrices` *Environment Matrices*

Description

Provides the frequencies of each amino acid within the environment.

Usage

```
env.matrices(env)
```

Arguments

`env`	a character string vector containing the environments.

Value

Returns a list of two dataframes. The first, shown the environment in matrix form. The second provides the frequencies of each amino acid within the environments.

Author(s)

Juan Carlos Aledo

References

Aledo et al. Sci Rep. 2015; 5: 16955. (PMID: 26597773)

See Also

env.extract(), env.Ztest() and env.plot()

Examples

```
env.matrices(c('ANQRmCTPQ', 'LYPPmQTPC', 'XXGSmSGXX'))
```

env.plot *Differential Sequence Environment Plot*

Description

Plots the Z statistics at each position within the environment for the requested amino acid.

Usage

```
env.plot(Z, aa, pValue = 0.001, ylim = c(-8,6), ty = 'p',title = "")
```

Arguments

`Z`	a matrix containing the standardized difference in frequencies (positive - control).
`aa`	the amino acid of interest.
`pValue`	the p-Value chosen to confer statistical significance.
`ylim`	range of the dependent variable. If we pass the argument 'automatic', the function will choose a suitable range for you.
`ty`	what type of plot should be drawn ("p": points, "l": lines, "b": both).
`title`	character string giving a title for the plot.

Details

The p-Value is used to draw two horizontal lines delimiting the region supporting the null hypothesis: no significant differences. Points laying above or below of these lines cannot be explained by randomness.

Value

This function returns a plot for the requested amino acid.

Author(s)

Juan Carlos Aledo

References

Aledo et al. Sci Rep. 2015; 5: 16955. (PMID: 26597773)

See Also

env.extract(), env.matrices() and env.Ztest()

Examples

```
## Not run: ## Get the matrices
pos = env.matrices(hmeto$positive)[[2]][,-11]
ctr = env.matrices(hmeto$control)[[2]][,-11]
## Run the test
Z = env.Ztest(pos, ctr, alpha = 0.0001)[[1]]
## Plot the results
env.plot(Z, aa = 'E', pValue = 0.05)
## End(Not run)
```

---

env.Ztest *Preferred/Avoided Amino Acids Within an Environment*

Description

Searches for amino acids either overrepresented or avoided at given positions within a sequence environment.

Usage

```
env.Ztest(pos, ctr, alpha = 0.05)
```

Arguments

pos	a 21 x m matrix containing the absolute frequencies of 21 amino acids at the m positions, in the positive environments.
ctr	a 21 x m matrix containing the absolute frequencies of 21 amino acids at the m positions, in the control environments.
alpha	significance level.

Details

Please, note that in addition to the 20 proteinogenetic amino acid we are using the symbol X when the target (central) residue is closer to the N-terminal or C-terminal of the protein than the radius used.

Value

Returns a list with three elements: (1) a matrix with the values of the Z statistical. (2) A dataframe with information regarding amino acid overrepresented in the positive environments, and (3) a dataframe similar to the previous one, but for amino acids avoided from the positive environments.

Author(s)

Juan Carlos Aledo

References

Aledo et al. Sci Rep. 2015; 5: 16955. (PMID: 26597773)

See Also

env.extract(), env.matrices() and env.plot()

Examples

```
pos = env.matrices(hmeto$positive)[[2]][,-11]; ctr = env.matrices(hmeto$control)[[2]][,-11]
env.Ztest(pos, ctr, alpha = 0.0001)
```

`find.aaindex` *Find the Amino Acid Indexes*

Description

Finds amino acid indexes.

Usage

```
find.aaindex(word)
```

Arguments

`word`	a character string for the key-word of interest.

Value

The number and ID of the indexes matching the requested word

Author(s)

Juan Carlos Aledo

References

https://www.genome.jp/aaindex/AAindex/list_of_indices

See Also

ptm.plot()

Examples

```
find.aaindex('mutability')
find.aaindex('Kyte-Doolittle')
```

`foldx.assembly` *Compute Assembly Free Energy*

Description

Computes changes in the Gibbs free energy of the assembly process of a protein.

Usage

```
foldx.assembly(pdb, mol1, mol2, pH = 7, I = 0.05)
```

Arguments

`pdb`	the 4-letter identifier of a PDB structure or the path to a PDB file.
`mol1`	molecule or group of molecules interacting with mol2 (see details)
`mol2`	molecule or group of molecules interacting with mol1 (see details)
`pH`	a numeric value between 0 and 14.
`I`	a value indicating the molar ionic strength.

Details

This function implements the FoldX's command 'AnalyseComplex', which allows to determine the interaction energy between two molecules or two groups of molecules. For instance, if in a dimeric protein, formed by chain A and B, we may set: mol1 = 'A', mol2 = 'B'. If we are dealing with a trimer, we may set: mol1 = 'A', mol2: 'AB'.

Value

The function returns a dataframe with the residues that make up the interface between mol1 and mol2, as well as the change in Gibbs free energy, DG, of the assembly process for the requested subunits.

Author(s)

Juan Carlos Aledo

References

Schymkowitz et al (2005) Nucl. Ac. Res. 33:W382-W388.

See Also

foldx.stab()

Examples

```
## Not run: foldx.assembly(pdb = '1sev', mol1 = 'A', mol2 = 'B')
```

`foldx.mut` *Compute Changes in Stability (DDG)*

Description

Computes changes in the stability of a protein after a residue mutation using a force-field approach.

Usage

```
foldx.mut(pdb, ch, pos, newres = "", pH =7, method = "buildmodel", keepfiles = FALSE)
```

Arguments

`pdb`	the 4-letter identifier of a PDB structure or the path to a PDB file.
`ch`	a letter identifying the chain of interest.
`pos`	the position, in the primary structure, of the residue to be mutated.
`newres`	the one letter code of the residue to be incorporated. When a value is not entered for this parameter, then the function will compute DDG for the mutation to any possible amino acid (including phosphoserine, phosphothreonine, phosphotyrosine and hydroxiproline in the case of the 'positionscan' method).
`pH`	a numeric value between 0 and 14.
`method`	a character string specifying the approach to be used; it must be one of 'buildmodel', 'positionscan'.
`keepfiles`	logical, when TRUE the repaired PDB file is saved in the working directory.

Details

Two computational approaches for prediction of the effect of amino acid changes on protein stability are implemented. FoldX (buildmodel and positionscan methods) uses a force field approach and although it has been proved to be satisfactorily accurate, it is also a time-consuming method. An alternative much faster is I-Mutant, a method base on machine-learning

Value

The function computes and returns the DDG (kcal/mol) for the requested residue change, defined as DDG = DGmt - DGwt, where DG is the Gibbs free energy for the folding of the protein from its unfolded state. Thus, a positive value means a destabilizing effect, and vice versa.

Author(s)

Juan Carlos Aledo

References

Schymkowitz et al (2005) Nucl. Ac. Res. 33:W382-W388.

See Also

imutant(), ddG.profile()

Examples

```
## Not run: foldx.mut('1aaq', 'A', 45, 'R')
```

---

foldx.stab *Compute Folding Free Energy (DG)*

Description

Computes changes in the Gibbs free energy of the folding process of a protein.

Usage

```
foldx.stab(pdb, pH = 7, I = 0.05)
```

Arguments

pdb	the 4-letter identifier of a PDB structure or the path to a PDB file.
pH	a numeric value between 0 and 14.
I	a value indicating the molar ionic strength.

Details

This function implements the FoldX's command 'Stability'

Value

The function computes and returns the DG (kcal/mol) of the folding process of the requested protein.

Author(s)

Juan Carlos Aledo

References

Schymkowitz et al (2005) Nucl. Ac. Res. 33:W382-W388.

See Also

foldx.assembly()

Examples

```
## Not run: foldx.stab('5zok')
```

`get.area` *Atomic Solvation Energies.*

Description

Computes online surface energies using the Getarea server.

Usage

```
get.area(pdb, keepfiles = FALSE)
```

Arguments

`pdb`	is either a PDB id, or the path to a pdb file.
`keepfiles`	logical, if TRUE the dataframe will be saved in the working directory and we will keep the getarea txt file.

Details

If the option keepfiles is set as TRUE, then txt and Rda files are saved in the working directory.

Value

This function returns a dataframe containing the requested information.

Author(s)

Juan Carlos Aledo

References

Fraczkiewicz, R. and Braun, W. (1998) J. Comp. Chem., 19, 319-333.

See Also

compute.dssp(), atom.dpx(), res.dpx(), acc.dssp(), str.part()

Examples

```
## Not run: get.area('3cwm')
```

get.go *Get Gene Ontology Annotation*

Description

Gets the gene ontology annotations for a given protein.

Usage

```
get.go(id, format = 'dataframe', silent = FALSE)
```

Arguments

`id`	the UniProt identifier of the protein of interest.
`format`	string indicating the output's format. It should be either 'dataframe' or 'string'. The 'string' format may be convenient when subsequent GO terms enrichment analysis is intended.
`silent`	logical, if FALSE print details of the reading process.

Value

Returns a dataframe (by default) with GO IDs linked to the protein of interest, as well as additional information related to these GO ids. A string with the GO ids can be obtained as output if indicated by means of the argument 'format'.

Author(s)

Juan Carlos Aledo

References

Rhee et al. (2008) Nature Reviews Genetics 9:509–515.

See Also

search.go, term.go(), bg.go(), hdfisher.go(), gorilla(), net.go()

Examples

```
## Not run: get.go('P01009')
```

get.seq — *Import a Protein Sequence from a Database*

Description

Imports a protein sequence from a selected database.

Usage

```
get.seq(id, db = 'uniprot', as.string = TRUE)
```

Arguments

`id`	the identifier of the protein of interest.
`db`	a character string specifying the desired database; it must be one of 'uniprot', 'metosite', 'pdb', 'kegg-aa', 'kegg-nt'.
`as.string`	logical, if TRUE the imported sequence will be returned as a character string.

Details

MetOSite uses the same type of protein ID than UniProt. However, if the chosen database is PDB, the identifier should be the 4-character unique identifier characteristic of PDB, followed by colon and the chain of interest. For instance, '2OCC:B' means we are interested in the sequence of chain B from the structure 2OCC. KEGG used its own IDs (see examples).

Value

Returns a protein (or nucleotide) sequence either as a character vector or a as a character string.

Author(s)

Juan Carlos Aledo

Examples

```
get.seq('P01009')
```

`gl.scan` *Scan a Protein in Search of OGlcNAc Sites*

Description

Scans the indicated protein in search of glycosylation sites.

Usage

```
gl.scan(up_id, db = 'all')
```

Arguments

`up_id`	a character string corresponding to the UniProt ID.
`db`	the database where to search. It should be one among 'PSP', 'dbPTM', 'all'.

Details

If db = 'all' has been selected, it may happen that the same residue appears in several rows if it is present in different databases.

Value

Returns a dataframe where each row corresponds to a modifiable residue.

Author(s)

Juan Carlos Aledo

References

Hornbeck et al. Nucleic Acids Res. 2019 47:D433-D441, (PMID: 30445427).

Huang et al. Nucleic Acids Res. 2019 47:D298-D308, (PMID: 30418626).

See Also

meto.scan(), ac.scan(), me.scan(), ub.scan(), su.scan(), p.scan(), sni.scan(), ni.scan(), ptm.scan(), reg.scan(), dis.scan()

Examples

```
## Not run: gl.scan('P08670', db = 'PSP')
```

`gracefully_fail` *Check that Internet Resource Work Properly and Fail Gracefully When Not*

Description

Checks that internet resource works properly and fail gracefully when not.

Usage

```
gracefully_fail(call, timeout = 10,  ...)
```

Arguments

`call`	url of the resource.
`timeout`	set maximum request time in seconds.
`...`	further named parameters, such as query, headers, etc.

Details

To be used as an ancillary function.

Value

The response object or NULL when the server does not respond properly.

Author(s)

thefactmachine

References

https://gist.github.com/thefactmachine/18279b7796c0836d9188

Examples

```
gracefully_fail("http://httpbin.org/delay/2")
```

hdfisher.go *Hypothesis-Driven Fisher Test*

Description

Carries out an enrichment Fisher's test using a hypothesis driven approach.

Usage

```
hdfisher.go(target, background, query, analysis = 'enrichment')
```

Arguments

`target`	either a vector containing the UniProt IDs of the target set or the path to the txt file containing the list of IDs.
`background`	a dataframe with two columns (Uniprot ID and GO terms) and as many rows as different proteins there are in the background set.
`query`	character string defining the query.
`analysis`	a character string indicating whether the desired analysis is the enrichment ('enrichment') or depletion ('depletion').

Value

Returns a list that contains the contingency table and the p-Value.

Author(s)

Juan Carlos Aledo

References

Rhee et al. (2008) Nature Reviews Genetics 9:509–515.

See Also

search.go(), term.go(), get.go(), bg.go(), go.enrich(), gorilla(), net.go()

Examples

```
## Not run: hdfisher.go(c('Q14667', 'Q5JSZ5'), bg.go(c('Q14667', 'Q5JSZ5', 'P13196')), 'ion')
```

hmeto *Human MetO sites oxidized by hydrogen peroxide treatment.*

Description

A dataset containing data regarding human MetO sites oxidized by H2O2.

Usage

```
hmeto
```

Format

A data frame with 4472 rows and 15 variables:

prot_id UniProt ID of the oxidized protein

prot_name the protein's name

met_pos the position of the MetO site in the primary structure

met_vivo_vitro conditions under which the oxidation experiment was carried out

MetOsites array with all the sites oxidized in that protein

site_id primary key identifying the site

positive sequence environment of the MetO site

control sequence environment of a non oxidized Met from the same protein

IDP Intrinsically Disordered Proteins, 0: the protein is not found in DisProt; 1: the protein contains disordered regions; 2: the protein may contain disordered regions but the experimental evidences are ambiguous

IDR Intrinsically Disordered Region, TRUE: the MetO site belong to the IDR, FALSE: the MetO site doesn't belong to the IDR

abundance protein abundance, in ppm

N protein length, in number of residues

met number of methionine residues

fmet relative frequency of Met in that protein

prot_vivo_vitro whether the protein has been described to be oxidized in vivo, in vitro or under both conditions

Source

https://metosite.uma.es/

id.features — *Features Related to the Protein Entry*

Description

Obtains features related to the provided id.

Usage

```
id.features(id, features = "")
```

Arguments

`id`	the UniProt identifier of the protein of interest.
`features`	a string identifying the features (comma separated) to be recovered.

Details

By default the the function provides info regarding the following features: id, reviewed, entry name and organism. If wished, this list of features can be expanded using the argument 'features'. There is a large list of features that can be retrieved. You can look up your relevant feature's name in the full list of UniProtKB found at https://www.uniprot.org/help/return_fields.

Value

Returns a named list with the requested features.

Author(s)

Juan Carlos Aledo

See Also

species.mapping()

Examples

```
## Not run: id.features('P04406', features = 'ec,keyword,xref_pdb')
```

id.mapping — *Identifier Mapping*

Description

Mapping between protein identifiers.

Usage

```
id.mapping(id, from, to)
```

Arguments

id	the identifier to be converted.
from	the type for the identifier of origin; it must be one of 'uniprot', 'pdb', or 'kegg'.
to	the type for the identifier of destination; it must be one of 'uniprot', 'pdb', or 'kegg'.

Value

Returns a character string corresponding to the requested identifier.

Author(s)

Juan Carlos Aledo

See Also

pdb2uniprot(), uniprot2pdb()

Examples

```
## Not run: id.mapping('P01009', from = 'uniprot', to = 'pdb')
```

imutant — *Compute Changes in Stability (DDG)*

Description

Computes changes in the stability of a protein after a residue mutation using a machine-learning approach.

Usage

```
imutant(protein, ch = "_", pos, newres = "", pH = 7, Te = 25, timeout = 60)
```

Arguments

`protein`	either the 4-letter identifier of a PDB structure, or the amino acid sequence (one letter amino acid code) of a protein.
`ch`	a letter identifying the chain of interest.
`pos`	the position, in the primary structure, of the residue to be mutated.
`newres`	the one letter code of the residue to be incorporated. When a value is not entered for this parameter, then the function will compute DDG for the mutation to any possible amino acid.
`pH`	a numeric value between 0 and 14.
`Te`	a numeric value indicating the temperature in degrees Celsius.
`timeout`	maximum time to wait, in seconds, for a response from the I-Mutant server.

Details

This function implements the I-Mutant v2.0 tool, which is a fast method based on a support vector machine approach to predict protein stability changes upon single point mutations.

Value

The function computes and returns a dataframe containing the following variables:

- Position: Position in the primary structure of the mutated residue.
- WT: Amino acid found at that position in the wild-type protein.
- NW: New amino acid found in the mutated protein.
- DDG: Change in Gibbs free energy (kcal/mol), defined as DDG = DGmt - DGwt, where DG is the change in Gibbs free energy for the folding of the protein from its unfolded state. Thus, a positive value means a stabilizing effect, and vice versa.
- pH: -log H+]
- T: Temperature in Celsius degrees.
- RSA: Relative Solvent Accessible Area (Only if a PDB file has been provided).

Author(s)

Juan Carlos Aledo

References

Capriotti et al (2005) Nucl. Ac. Res. 33:W306-W310.

See Also

foldx.mut(), ddG.profile()

Examples

```
## Not run: imutant(protein = '1u8f', ch = 'O', pos = 46, newres = 'K')
```

is.at *Check Residue a Fixed Position*

Description

Checks if a given amino acid is at a given position.

Usage

```
is.at(at, target, aa = 'M', uniprot = TRUE)
```

Arguments

at	the position in the primary structure of the protein.
target	a character string specifying the UniProt ID of the protein of interest or, alternatively, the sequence of that protein.
aa	the amino acid of interest.
uniprot	logical, if TRUE the argument 'target' should be an ID.

Details

Please, note that when uniprot is set to FALSE, target can be the string returned by a suitable function, such as get.seq or other.

Value

Returns a boolean. Either the residue is present at that position or not.

Author(s)

Juan Carlos Aledo

See Also

aa.at(), renum.pdb(), renum.meto(), renum(), aa.comp()

Examples

```
## Not run: is.at(28, 'P01009', 'Q')
```

kegg.uniprot — *Identifier Mapping From KEGG to UniProt*

Description

Mapping between KEGG and UniProt protein identifiers.

Usage

```
kegg.uniprot(id)
```

Arguments

id	the identifier to be converted.

Value

Returns a character string corresponding to the requested identifier.

Author(s)

Juan Carlos Aledo

See Also

id.mapping()

Examples

```
## Not run: kegg.uniprot('hsa:5265')
```

me.scan — *Scan a Protein in Search of Methylation Sites*

Description

Scans the indicated protein in search of methylation sites.

Usage

```
me.scan(up_id, db = 'all')
```

Arguments

up_id	a character string corresponding to the UniProt ID.
db	the database where to search. It should be one among 'PSP', 'dbPTM', 'all'.

Details

If db = 'all' has been selected, it may happen that the same residue appears in several rows if it is present in different databases.

Value

Returns a dataframe where each row corresponds to a modifiable residue.

Author(s)

Juan Carlos Aledo

References

Hornbeck et al. Nucleic Acids Res. 2019 47:D433-D441, (PMID: 30445427).

Huang et al. Nucleic Acids Res. 2019 47:D298-D308, (PMID: 30418626).

See Also

meto.scan(), ac.scan(), p.scan(), ub.scan(), su.scan(), gl.scan(), sni.scan(), ni.scan(), ptm.scan(), reg.scan(), dis.scan()

Examples

```
## Not run: me.scan('Q16695', db = 'PSP')
```

---

meto.list — *List Proteins Found in MetOSite Matching a Keyword*

Description

Lists proteins found in MetOSite with names matching the keyword.

Usage

```
meto.list(keyword)
```

Arguments

`keyword`	a character string corresponding to the keyword

Value

This function returns a dataframe with the uniprot id, the protein name and the species, for those proteins present into MetOSite whose name contains the keyword.

Author(s)

Juan Carlos Aledo

References

Valverde et al. 2019. Bioinformatics 35:4849-4850 (PMID: 31197322)

See Also

meto.search(), meto.scan()

Examples

```
meto.list('inhibitor')
```

---

meto.scan *Scans a Protein in Search of MetO Sites*

Description

Scans a given protein in search of MetO sites.

Usage

```
meto.scan(up_id, report = 1)
```

Arguments

up_id	a character string corresponding to the UniProt ID.
report	it should be a natural number between 1 and 3.

Details

When the 'report' parameter has been set to 1, this function returns a brief report providing the position, the function category and literature references concerning the residues detected as MetO, if any. If we wish to obtain a more detailed report, the option should be: report = 2. Finally, If we want a detailed and printable report (saved in the current directory), we should set report = 3

Value

This function returns a report regarding the MetO sites found, if any, in the protein of interest.

Author(s)

Juan Carlos Aledo

References

Valverde et al. 2019. Bioinformatics 35:4849-4850 (PMID: 31197322)

See Also

meto.search(), meto.list()

Examples

```
meto.scan('P01009')
```

---

meto.search *Search for Specific MetO Sites*

Description

Searches for specific MetO sites filtering MetOSite according to the selected criteria.

Usage

```
meto.search(highthroughput.group = TRUE,
                   bodyguard.group = TRUE,
                   regulatory.group = TRUE,
                   gain.activity = 2, loss.activity = 2, gain.ppi = 2,
                   loss.ppi = 2, change.stability = 2, change.location = 2,
                   organism = -1, oxidant = -1)
```

Arguments

`highthroughput.group`
: logical, when FALSE the sites described in a high-throughput study (unknown effect) are filtered out.

`bodyguard.group`
: logical, when FALSE the sites postulated to function as ROS sink (because when oxidized no apparent effect can be detected) are filtered out.

`regulatory.group`
: logical, when FALSE the sites whose oxidation affect the properties of the protein (and therefore may be involved in regulation) are filtered out.

`gain.activity`
: introduce 1 or 0 to indicate whether the oxidation of the selected sites implies a gain of activity or not, respectively. If we do not wish to use this property to filter, introduce 2.

`loss.activity`
: introduce 1 or 0 to indicate whether or not the oxidation of the selected sites implies a loss of activity or not, respectively. If we do not wish to use this property to filter, introduce 2.

`gain.ppi`
: introduce 1 or 0 to indicate whether the oxidation of the selected sites implies a gain of protein-protein interaction or not, respectively. If we do not wish to use this property to filter, introduce 2.

`loss.ppi` introduce 1 or 0 to indicate whether or not the oxidation of the selected sites implies a loss of protein-protein interaction or not, respectively. If we do not wish to use this property to filter, introduce 2.

`change.stability` introduce 1 or 0 to indicate whether the oxidation of the selected sites leads to a change in the protein stability or not, respectively. If we do not wish to use this property to filter, introduce 2.

`change.location` introduce 1 or 0 to indicate whether or not the oxidation of the selected sites implies a change of localization or not, respectively. If we do not wish to use this property to filter, introduce 2.

`organism` a character string indicating the scientific name of the species of interest, or -1 if we do not wish to filter by species.

`oxidant` a character string indicating the oxidant, or -1 if we do not wish to filter by oxidants.

Details

Note that all the arguments of this function are optional. We only pass an argument to the function when we want to use that parameter to filter. Thus, meto.search() will return all the MetO sites found in the database MetOSite.

Value

This function returns a dataframe with a line per MetO site.

Author(s)

Juan Carlos Aledo

References

Valverde et al. 2019. Bioinformatics 35:4849-4850 (PMID: 31197322)

See Also

meto.scan(), meto.list()

Examples

```
meto.search(organism = 'Homo sapiens', oxidant = 'HClO')
```

mkdssp *Compute DSSP File Using an In-House Version of the DSSP Software*

Description

Computes the DSSP file using an in-house version of the DSSP software.

Usage

```
mkdssp(pdb, method = 'ptm', exefile = "dssp")
```

Arguments

pdb	is either a 4-character identifier of the PDB structure, or the path to a pdb file.
method	a character string specifying the desired method to get the dssp dataframe; it should be one of 'ptm' or 'bio3d'.
exefile	file path to the DSSP executable on your system (i.e. how is DSSP invoked).

Details

The structure of the output data depends on the method chosen, but it will always contain the DSSP-related data.

Value

Returns either a dataframe containing the information extracted from the dssp file (method ptm), or a list with that information (method bio3d).

Author(s)

Juan Carlos Aledo

References

Touw et al (2015) Nucl. Ac. Res. 43(Database issue): D364-D368 (PMID: 25352545).

See Also

download.dssp(), parse.dssp(), compute.dssp() and acc.dssp()

Examples

```
## Not run: mkdssp('3cwm', method = 'ptm')
```

msa — *Multiple Sequence Alignment*

Description

Aligns multiple protein sequences.

Usage

```
msa(sequences, ids = names(sequences), sfile = FALSE, inhouse = FALSE)
```

Arguments

`sequences`	vector containing the sequences.
`ids`	vector containing the sequences' ids.
`sfile`	if different to FALSE, then it should be a string indicating the path to save a fasta alignment file.
`inhouse`	logical, if TRUE the in-house MUSCLE software is used. It must be installed on your system and in the search path for executables.

Value

Returns a list of four elements. The first one ($seq) provides the sequences analyzed, the second element ($ids) returns the identifiers, the third element ($aln) provides the alignment in fasta format and the fourth element ($ali) gives the alignment in matrix format.

References

Edgar RC. Nucl. Ac. Res. 2004 32:1792-1797.

Edgar RC. BMC Bioinformatics 5(1):113.

See Also

custom.aln(), list.hom(), parse.hssp(), get.hssp(), shannon()

Examples

```
## Not run: msa(sequences = sapply(c("P19446", "P40925", "P40926"), ptm::get.seq),
 ids = c("wmelon", "cyt", "mit"))
## End(Not run)
```

`net.go` *Gene Ontology Network*

Description

Explores the relationship among proteins from a given set.

Usage

```
net.go(data, threshold = 0.2, silent = FALSE)
```

Arguments

`data`	either a vector containing the UniProt IDs (vertices) or the path to the txt or rda file containing them.
`threshold`	threshold value of the Jaccard index above which two proteins are considered to be linked.
`silent`	logical, if FALSE print details of the running process.

Details

This function first searches the GO terms for each vertex and then computes the Jaccard index for each protein pair, based on their GO terms. Afterwards, an adjacency matrix is computed, where two proteins are linked if their Jaccard index is greater than the selected threshold.

Value

Returns a list containing (i) the dataframe corresponding to the computed Jaccard matrix, (ii) the adjacency matrix, (iii) a vector containing the vertices, and (iv) a matrix describing the edges of the network.

Author(s)

Pablo Aledo & Juan Carlos Aledo

References

Aledo & Aledo (2020) Antioxidants 9(10), 987.

Rhee et al. (2008) Nature Reviews Genetics 9:509–515.

See Also

search.go(), term.go(), get.go(), bg.go(), gorilla()

Examples

```
## Not run: net.go(path2data = "./GOvivo.txt")
```

ni.scan *Scan a Protein in Search of Nitration Sites*

Description

Scans the indicated protein in search of nitration sites.

Usage

```
ni.scan(up_id, db = 'all')
```

Arguments

up_id	a character string corresponding to the UniProt ID.
db	the database where to search. It should be one among 'PSP', 'dbPTM', 'all'.

Value

Returns a dataframe where each row corresponds to a modified residue.

Author(s)

Juan Carlos Aledo

References

Huang et al. Nucleic Acids Res. 2019 47:D298-D308, (PMID: 30418626).

See Also

meto.scan(), ac.scan(), me.scan(), ub.scan(), su.scan(), gl.scan(), sni.scan(), p.scan(), ptm.scan(), reg.scan(), dis.scan()

Examples

```
## Not run: ni.scan('P05202')
```

p.scan *Scan a Protein in Search of Phosphosites*

Description

Scans the indicated protein in search of phosphosites.

Usage

```
p.scan(up_id, db = 'all')
```

Arguments

up_id	a character string corresponding to the UniProt ID.
db	the database where to search. It should be one among 'PSP', 'dbPTM','dbPAF', 'PhosPhAt', 'Phospho.ELM', 'all'.

Details

If db = 'all' has been selected, it may happen that the same residue appears in several rows if it is present in different databases.

Value

Returns a dataframe where each row corresponds to a phosphorylatable residue.

Author(s)

Juan Carlos Aledo

References

Hornbeck et al. Nucleic Acids Res. 2019 47:D433-D441, (PMID: 30445427).

Huang et al. Nucleic Acids Res. 2019 47:D298-D308, (PMID: 30418626).

Ullah et al. Sci. Rep. 2016 6:23534, (PMID: 27010073).

Durek et al. Nucleic Acids Res.2010 38:D828-D834, (PMID: 19880383).

Dinkel et al. Nucleic Acids Res. 2011 39:D261-D567 (PMID: 21062810).

See Also

meto.scan(), ac.scan(), me.scan(), ub.scan(), su.scan(), gl.scan(), sni.scan(), ni.scan(), ptm.scan(), reg.scan(), dis.scan()

Examples

```
## Not run: p.scan('P01009', db = 'PSP')
```

pairwise.dist *Compute Euclidean Distances*

Description

Computes the pairwise distance matrix between two sets of points

Usage

```
pairwise.dist(a, b, squared = TRUE)
```

Arguments

a, b	matrices (NxD) and (MxD), respectively, where each row represents a D-dimensional point.
squared	return containing squared Euclidean distance

Value

Euclidean distance matrix (NxM). An attribute "squared" set to the value of param `squared` is provided.

Examples

```
pairwise.dist(matrix(1:9, ncol = 3), matrix(9:1, ncol = 3))
```

parse.dssp *Parse a DSSP File to Return a Dataframe*

Description

Parses a DSSP file to return a dataframe.

Usage

```
parse.dssp(file, keepfiles = FALSE)
```

Arguments

file	input dssp file.
keepfiles	logical, if TRUE the dataframe will be saved in the working directory and we will keep the dssp file.

Details

If the argument 'keepfiles' is not set to TRUE, the dssp file used to get the parsed dataframe will be removed.

Value

Returns a dataframe providing data for 'acc', 'ss', 'phi' and 'psi' for each residues from the structure.

Author(s)

Juan Carlos Aledo

References

Touw et al (2015) Nucl. Ac. Res. 43(Database issue): D364-D368 (PMID: 25352545).

See Also

download.dssp(), compute.dssp(), mkdssp() and acc.dssp()

Examples

```
## Not run: compute.dssp('3cwm'); parse.dssp('3cwm.dssp')
```

---

pdb.chain *Download and/or Split PDB Files.*

Description

Downloads a PDB file (if required) and splits it to provide a file by chain.

Usage

```
pdb.chain(pdb, keepfiles = FALSE)
```

Arguments

pdb	the path to the PDB of interest or a 4-letter identifier.
keepfiles	logical, if TRUE the function makes a 'temp' directory within the current directory and save in it a pdb file for each chain present in the given structure.

Value

The function returns a chr vector where each coordinate is a chain from the structure.

Author(s)

Juan Carlos Aledo

Examples

```
## Not run: pdb.chain('1bpl')
```

pdb.quaternary *Protein Subunit Composition*

Description

Determines the subunit composition of a given protein.

Usage

```
pdb.quaternary(pdb, keepfiles = FALSE)
```

Arguments

pdb	the path to the PDB of interest or a 4-letter identifier.
keepfiles	logical, if TRUE the fasta file containing the alignment of the subunits is saved in the current directory, as well as the split pdb files.

Details

A fasta file containing the alignment among the subunit sequences can be saved in the current directory if required.

Value

This function returns a list with four elements: (i) a distances matrix, (ii) the sequences, (iii) chains id, (iv) the PDB ID used.

Author(s)

Juan Carlos Aledo

Examples

```
## Not run: pdb.quaternary('1bpl')
```

pdb.select *Select the PDB with the Optimal Coverage to the UniProt Sequence*

Description

Select the PDB and chain with the optimal coverage to a given UniProt sequence.

Usage

```
pdb.select(up_id, threshold = 0.9)
```

Arguments

`up_id`	the UniProt ID.
`threshold`	coverage value that when reached the search is halted.

Value

A list of two elements: (i) the PDB ID and (ii) the chain. The coverage with the UniProt sequence is given as an attribute.

Author(s)

Juan Carlos Aledo

See Also

pdb.quaternary(), pdb.chain(), pdb.res(), pdb.pep(), uniprot2pdb(), pdb2uniprot()

Examples

```
## Not run: pdb.select('P01009', threshold = 0.8)
```

---

pdb.seq *Get Chain Sequences*

Description

Gets the sequences of the chain find in a given PDB.

Usage

```
pdb.seq(pdb)
```

Arguments

`pdb`	the 4-letter PDB identifier.

Value

Returns a dataframe with as many rows as different chains are present in the PDB. For each row six variables are returned: (i) the entry id, (ii) the entity id, (iii) the chain, (iv) the protein name, (v) the species and (vi) the sequence.

Author(s)

Juan Carlos Aledo

Examples

```
## Not run: pdb.seq('1bpl')
```

pdb.uniprot *Identifier Mapping From PDB to UniProt*

Description

Mapping between PDB and UniProt protein identifiers.

Usage

```
pdb.uniprot(id)
```

Arguments

id	the identifier to be converted.

Value

Returns a character string corresponding to the requested identifier.

Author(s)

Juan Carlos Aledo

See Also

id.mapping()

Examples

```
## Not run: pdb.uniprot('3cwm')
```

pdb2uniprot *Return the UniProt ID Given the PDB and Chain IDs*

Description

Returns the uniprot id of a given chain within a PDB structure.

Usage

```
pdb2uniprot(pdb, chain)
```

Arguments

pdb	the 4-letter PDB identifier.
chain	letter identifying the chain.

Value

The function returns the UniProt ID for the chain of interest.

Author(s)

Juan Carlos Aledo

See Also

uniprot2pdb(), id.mapping()

Examples

```
## Not run: pdb2uniprot('1u8f', 'O')
```

---

prot2codon *Find the Coding Triplets for a Given Protein*

Description

Finds the codons corresponding to a given protein.

Usage

```
prot2codon(prot, chain = "", laxity = TRUE)
```

Arguments

prot	is either a UniProt or PDB id, or the path to a pdb file.
chain	when prot corresponds to a pdb, the chain of interest must be provided.
laxity	logical, if FALSE the program stop when a mismatch between the protein and the gene sequences is detected. Otherwise the program doesn't stop and at the end points out the mismatches.

Value

Returns a dataframe with as many rows as residues has the protein.

Author(s)

Juan Carlos Aledo

Examples

```
## Not run: prot2codon('P01009')
```

ptm.plot *Plot Values of a Property and PTM Sites Along the Protein Sequence*

Description

Represents the values of a property and show the PTM sites along a protein sequence.

Usage

```
ptm.plot(up_id, pdb = "", property, ptm, window = 1, sdata = FALSE, ...)
```

Arguments

`up_id`	a character string for the UniProt ID of the protein of interest.
`pdb`	Optional argument to indicate the PDB and chain to be used (i.e. '1u8f.O'). If we leave this argument empty, the function will make the election for us whenever possible.
`property`	a character string indicating the property of interest. It should be one of 'sasa', 'acc', 'dpx', 'eiip', 'volume', 'polarizability', 'avg.hyd', 'pi.hel', 'a.hel', 'b.sheet', 'B.factor', or 'own'.
`ptm`	a character vector indicating the PTMs of interest. It should be among: 'ac' (acetylation), 'me' (methylation), 'meto' (sulfoxidation), 'p' (phosphorylation), 'ni' (nitration), 'su' (sumoylation) or 'ub' (ubiquitination), 'gl' (glycosylation), 'sni' (S-nitrosylation),'reg' (regulatory), 'dis' (disease).
`window`	positive integer indicating the window size for smoothing with a sliding window average (default: 1, i.e. no smoothing).
`sdata`	logical, if TRUE save a Rda file with the relevant data in the current directory.
`...`	when the user want to use his/her own amino acid index, it can be passed as a named vector.

Details

If the property 'own' is selected, a named vector with the own index for the 20 amino acids should be passed as argument. Currently the supported properties are:

- sasa: Solvent-accessible surface area (3D)
- acc: Accessibility (3D)
- dpx: Depth (3D)
- volume: Normalized van der Waals volume (1D)
- mutability: Relative mutability, Jones 1992, (1D)
- helix: Average relative probability of helix, Kanehisa-Tsong 1980,(1D)
- beta-sheet: Average relative probability of beta-sheet, Kanehisa-Tsong 1980, (1D)
- pi-helix: Propensity of amino acids within pi-helices, Fodje-Al-Karadaghi 2002, (1D)

- hydropathy: Hydropathy index, Kyte-Doolittle 1982, (1D)
- avg.hyd: Normalized average hydrophobicity scales, Cid et al 1992, (1D)
- hplc: Retention coefficient in HPLC at pH7.4, Meek 1980, (1D)
- argos: Hydrophobicity index, Argos et al 1982, (1D)
- eiip: Electron-ion interaction potential, Veljkovic et al 1985, (1D)
- polarizability: Polarizability parameter, Charton-Charton 1982, (1D)

For 3D properties such as sasa, acc or dpx, for which different values can be obtained depending on the quaternary structure, we first compute the property values for each residue in the whole protein and plotted them against the residue position. Then, the value for this property is computed in the isolated chain (a single polypeptide chain) and in a second plot, the differences between the values in the whole protein and the chain are plotted against the residue position.

Value

This function returns either one or two plots related to the chosen property along the primary structure, as well as the computed data if sdata has been set to TRUE.

Author(s)

Juan Carlos Aledo

See Also

find.aaindex()

Examples

```
## Not run: ptm.plot('P04406', property = 'sasa', window = 10, ptm = 'meto')
## Not run: ptm.ptm('P04406', property = 'dpx', ptm = c('meto', 'p'))
```

`ptm.scan` *Scan a Protein in Search of PTM Sites*

Description

Scans the indicated protein in search of PTM sites.

Usage

```
ptm.scan(up_id, renumerate = TRUE)
```

Arguments

`up_id`	a character string corresponding to the UniProt ID.
`renumerate`	logical, when TRUE the sequence numeration of MetO sites is that given by Uniprot, which may not coincide with that from MetOSite.

Details

The numerations of the sequences given by UniProt and MetOSite may or may not match. Sometimes one of the sequences corresponds to the precursor protein and the other to the processed mature protein.

Value

Returns a dataframe where each row corresponds to a residue, and the columns inform about the modifications.

Author(s)

Juan Carlos Aledo

References

Hornbeck et al. Nucleic Acids Res. 2019 47:D433-D441, (PMID: 30445427).

Huang et al. Nucleic Acids Res. 2019 47:D298-D308, (PMID: 30418626).

Ullah et al. Sci. Rep. 2016 6:23534, (PMID: 27010073).

Durek et al. Nucleic Acids Res.2010 38:D828-D834, (PMID: 19880383).

Dinkel et al. Nucleic Acids Res. 2011 39:D261-D567 (PMID: 21062810).

See Also

meto.scan(), ac.scan(), me.scan(), ub.scan(), su.scan(), gl.scan(), sni.scan(), ni.scan(), p.scan(), reg.scan(), dis.scan()

Examples

```
## Not run: ptm.scan('P01009', renumerate = TRUE)
```

---

reg.scan — *Scan a Protein in Search of Regulatory PTM Sites*

Description

Scans the indicated protein in search of regulatory PTM sites.

Usage

```
reg.scan(up_id)
```

Arguments

`up_id`	a character string corresponding to the UniProt ID.

Value

Returns a dataframe where each row corresponds to a residue, and the columns inform about the regulatory modifications.

Author(s)

Juan Carlos Aledo

References

Hornbeck et al. Nucleic Acids Res. 2019 47:D433-D441, (PMID: 30445427).

See Also

meto.scan(), ac.scan(), me.scan(), ub.scan(), su.scan(), gl.scan(), sni.scan(), ni.scan(), ptm.scan(), p.scan(), dis.scan()

Examples

```
## Not run: reg.scan('P01009')
```

`renum` *Renumerate Residue Position*

Description

Renumerates residue position.

Usage

```
renum(up_id, pos, from, to, ...)
```

Arguments

`up_id`	the UniProt ID.
`pos`	position in the initial sequence.
`from`	origin of the initial sequence, it should be one among 'uniprot', 'metosite' and 'pdb'.
`to`	target sequence, it should be one among 'uniprot', 'metosite' and 'pdb'.
`...`	additional arguments (PDB ID and chain) when 'pdb' is either origin or destination.

Details

Either the origin sequence or the target sequence should be uniprot. Nevertheless, the conversion pdb -> metosite, for instance, can be achieved through the path: pdb -> uniprot -> metosite. If 'pdb' is selected, then the PDB ID and the involved chain must be provided, in that order.

Value

Returns the final position.

Author(s)

Juan Carlos Aledo

See Also

is.at(), aa.at(), renum.pdb(), renum.meto(), aa.comp()

Examples

```
## Not run: renum(up_id = 'P01009', pos = 60, from = 'uniprot',
                          to = 'pdb', pdb = '1ATU', chain = 'A')
## End(Not run)
```

---

renum.meto *Renumerate Residue Position*

Description

Renumerates residue position of a MetOSite sequence to match the corresponding UniProt sequence

Usage

```
renum.meto(uniprot)
```

Arguments

`uniprot`	the UniProt ID.

Value

Returns a dataframe containing the re-numerated sequence.

Author(s)

Juan Carlos Aledo

See Also

is.at(), aa.at(), renum.pdb(), renum(), aa.comp()

Examples

```
## Not run: renum.meto('P01009')
```

renum.pdb *Renumerate Residue Position*

Description

Renumerates residue position of a PDB sequence to match the corresponding UniProt sequence.

Usage

```
renum.pdb(pdb, chain, uniprot)
```

Arguments

pdb	the PDB ID or the path to a pdb file.
chain	the chain of interest.
uniprot	the UniProt ID.

Value

Returns a dataframe containing the re-numerated sequence.

Author(s)

Juan Carlos Aledo

See Also

is.at(), aa.at(), renum.meto(), renum(), aa.compo()

Examples

```
## Not run: renum.pdb(pdb = '121P', chain = 'A', uniprot = 'P01112')
```

res.dpx *Residue Depth Analysis*

Description

Computes the depth from the surface for each protein's residue.

Usage

```
res.dpx(pdb, aa = 'all')
```

Arguments

pdb	is either a PDB id, or the path to a pdb file.
aa	one letter code for the amino acid of interest, or 'all' for all the protein residues.

Details

This function computes the depth, defined as the distance in angstroms between the residue and the closest atom on the protein surface.

Value

A dataframe with the computed depths.

Author(s)

Juan Carlos Aledo

References

Pintar et al. 2003. Bioinformatics 19:313-314 (PMID: 12538266)

See Also

atom.dpx(), acc.dssp(), str.part()

Examples

```
## Not run: res.dpx('1cll')
```

---

saro.dist *Compute Distances to the Closest Aromatic Residues*

Description

Computes distances to the closest aromatic residues.

Usage

```
saro.dist(pdb, threshold = 7, rawdata = FALSE)
```

Arguments

pdb	either the path to the PDB file of interest or the 4-letters identifier.
threshold	distance in ångströms, between the S atom and the aromatic ring centroid, used as threshold.
rawdata	logical to indicate whether we also want the raw distance matrix between delta S and aromatic ring centroids.

Details

For each methionyl residue this function computes the distances to the closest aromatic ring from Y, F and W. When that distance is equal or lower to the threshold, it will be computed as a S-aromatic motif.

Value

The function returns a dataframe with as many rows as methionyl residues are found in the protein. The distances in ångströms to the closest tyrosine, phenylalanine and triptophan are given in the columns, as well as the number of S-aromatic motifs detected with each of these amino acids. Also a raw distance matrix can be provided.

Author(s)

Juan Carlos Aledo

References

Reid, Lindley & Thornton, FEBS Lett. 1985, 190:209-213.

See Also

saro.motif(), saro.geometry()

Examples

```
## Not run: saro.dist('1CLL')
```

---

saro.geometry *Compute Geometric Parameters of S-Aromatic Motifs*

Description

Computes distances and angles of S-aromatic motifs.

Usage

```
saro.geometry(pdb, rA, chainA = 'A', rB, chainB = 'A')
```

Arguments

`pdb`	either the path to the PDB file of interest or the 4-letters identifier.
`rA`	numeric position of one of the two residues involved in the motif.
`chainA`	a character indicating the chain to which belong the first residue.
`rB`	numeric position of the second residue involved in the motif.
`chainB`	a character indicating the chain to which belong the second residue.

Details

The distance between the delta sulfur atom and the centroid of the aromatic ring is computed, as well as the angle between this vector and the one perpendicular to the plane containing the aromatic ring. Based on the distance (d) and the angle (theta) the user decide whether the two residues are considered to be S-bonded or not (usually when $d < 7$ and $theta < 60º$).

Value

The function returns a dataframe providing the coordinates of the sulfur atom and the centroid (centroids when the aromatic residue is tryptophan), as well as the distance (ångströms) and the angle (degrees) mentioned above.

Author(s)

Juan Carlos Aledo

References

Reid, Lindley & Thornton, FEBS Lett. 1985, 190, 209-213.

See Also

saro.motif(), saro.dist()

Examples

```
## Not run: saro.geometry('1CLL', rA = 141, rB = 145)
```

`saro.motif` *Search for S-Aromatic Motifs*

Description

Searches for S-aromatic motifs in proteins.

Usage

```
saro.motif(pdb, threshold = 7, onlySaro = TRUE)
```

Arguments

`pdb`	either the path to the PDB file of interest or the 4-letters identifier.
`threshold`	distance in ångströms, between the S atom and the aromatic ring centroid, used as threshold.
`onlySaro`	logical, if FALSE the output includes information about Met residues that are not involved in S-aromatic motifs.

Details

For each methionyl residue taking place in a S-aromatic motif, this function computes the aromatic residues involved, the distance between the delta sulfur and the aromatic ring's centroid, as well as the angle between the sulfur-aromatic vector and the normal vector of the plane containing the aromatic ring.

Value

The function returns a dataframe reporting the S-aromatic motifs found for the protein of interest.

Author(s)

Juan Carlos Aledo

References

Reid, Lindley & Thornton, FEBS Lett. 1985, 190, 209-213.

See Also

saro.dist(), saro.geometry()

Examples

```
## Not run: saro.motif('1CLL')
```

---

search.go *Search a Simple User Query*

Description

Searches a simple user query.

Usage

```
search.go(query)
```

Arguments

`query`	character string defining the query.

Value

Returns a dataframe containing the GO IDs found associated to the query, as well as other information related to these terms.

Author(s)

Juan Carlos Aledo

References

Rhee et al. (2008) Nature Reviews Genetics 9:509–515.

See Also

term.go(), get.go(), bg.go(), hdfisher.go(), gorilla(), net.go()

Examples

```
## Not run: search.go('oxidative stress')
```

`sni.scan` *Scan a Protein in Search of S-nitrosylation Sites*

Description

Scans the indicated protein in search of S-nitrosylation sites.

Usage

```
sni.scan(up_id, db = 'all')
```

Arguments

up_id	a character string corresponding to the UniProt ID.
db	the database where to search. It should be one among 'PSP', 'dbPTM', 'all'.

Value

Returns a dataframe where each row corresponds to a modifiable residue.

Author(s)

Juan Carlos Aledo

References

Huang et al. Nucleic Acids Res. 2019 47:D298-D308, (PMID: 30418626).

See Also

meto.scan(), ac.scan(), me.scan(), ub.scan(), su.scan(), gl.scan(), p.scan(), ni.scan(), ptm.scan(), reg.scan(), dis.scan()

Examples

```
## Not run: sni.scan('P01009')
```

species.kegg — *Convert Between Species Name and KEGG 3-Letter Code Format*

Description

Converts between species name and KEGG 3-letter code format.

Usage

```
species.kegg(organism, from = 'scientific')
```

Arguments

`organism`	character string defining the organisms.
`from`	string indicating the character of the provided name. It should be one of 'vulgar', 'scientific', '3-letter'.

Value

Returns a dataframe with the entries matching the request.

Author(s)

Juan Carlos Aledo

See Also

id.features(), species.mapping()

Examples

```
## Not run: species.kegg('chempanzee', from = 'vulgar')
## Not run: species.kegg('Pan paniscus')
## Not run: species.kegg('ppo', from = '3-letter')
```

species.mapping — *Map Protein ID to Species*

Description

Maps a protein ID to its corresponding organism.

Usage

```
species.mapping(id, db = 'uniprot')
```

Arguments

`id`	the identifier of the protein of interest.
`db`	a character string specifying the corresponding database. Currently, only 'uniprot' or 'pdb' are valid options.

Value

Returns a character string identifying the organism to which the given protein belong.

Author(s)

Juan Carlos Aledo

See Also

id.features()

Examples

```
## Not run: species.mapping('P01009')
```

`stru.part` *Partition of Structural Regions*

Description

Carries out a partition of the structural regions of a given protein.

Usage

```
stru.part(pdb, cutoff = 0.25)
```

Arguments

`pdb`	is either a PDB id, or the path to a pdb file
`cutoff`	accessibility below which a residue is considered to be buried.

Details

The accessibilities of a residue computed in the complex (ACCc) and in the monomer (ACCm) allow to distinguish four structural regions as follows.

Interior: ACCc < cutoff & (ACCm - ACCc) = 0.

Surface: ACCc > cutoff & (ACCm - ACCc) = 0.

Support: ACCm < cutoff & (ACCm - ACCc) > 0.

Rim: ACCc > cutoff & (ACCm - ACCc) > 0.

Core: ACCm > cutoff & ACCc < cutoff.

Value

A dataframe where each residue is assigned to one of the four structural groups considered.

Author(s)

Juan Carlos Aledo

References

Levy (2010) J. Mol. Biol. 403: 660-670 (PMID: 20868694).

See Also

atom.dpx(), res.dpx(), acc.dssp()

Examples

```
## Not run: stru.part('1u8f')
```

---

su.scan *Scan a Protein in Search of Sumoylation Sites*

Description

Scans the indicated protein in search of sumoylation sites.

Usage

```
su.scan(up_id, db = 'all')
```

Arguments

up_id	a character string corresponding to the UniProt ID.
db	the database where to search. It should be one among 'PSP', 'dbPTM', 'all'.

Details

If db = 'all' has been selected, it may happen that the same residue appears in several rows if it is present in different databases.

Value

Returns a dataframe where each row corresponds to a modifiable residue.

Author(s)

Juan Carlos Aledo

References

Hornbeck et al. Nucleic Acids Res. 2019 47:D433-D441, (PMID: 30445427).

Huang et al. Nucleic Acids Res. 2019 47:D298-D308, (PMID: 30418626).

See Also

meto.scan(), ac.scan(), me.scan(), ub.scan(), p.scan(), gl.scan(), sni.scan(), ni.scan(), ptm.scan(), reg.scan(), dis.scan()

Examples

```
## Not run: su.scan('Q16695', db = 'PSP')
```

---

term.go — *Get Core Information About the GO Term*

Description

Gets core information about the GO term of interest.

Usage

```
term.go(go, children = FALSE)
```

Arguments

`go`	GO id.
`children`	logical, when true GO children terms are returned.

Details

When the argument children is set to TRUE, the output of this function is a list with two elements: the first one is a dataframe with the core information, and the second one is a dataframe containing the children terms.

Value

Returns a dataframe containing core information such as term name and definition, reference, aspect, and whether or not the term is obsolete. If children is set to TRUE, the function returns a list.

Author(s)

Juan Carlos Aledo

References

Rhee et al. (2008) Nature Reviews Genetics 9:509–515.

See Also

search.go(), get.go(), bg.go(), hdfisher.go(), gorilla(), net.go()

Examples

```
## Not run: term.go('GO:0034599')
```

---

ub.scan *Scan a Protein in Search of Ubiquitination Sites*

Description

Scans the indicated protein in search of ubiquitination sites.

Usage

```
ub.scan(up_id, db = 'all')
```

Arguments

up_id	a character string corresponding to the UniProt ID.
db	the database where to search. It should be one among 'PSP', 'dbPTM', 'all'.

Details

If db = 'all' has been selected, it may happen that the same residue appears in several rows if it is present in different databases.

Value

Returns a dataframe where each row corresponds to a modifiable residue.

Author(s)

Juan Carlos Aledo

References

Hornbeck et al. Nucleic Acids Res. 2019 47:D433-D441, (PMID: 30445427).

Huang et al. Nucleic Acids Res. 2019 47:D298-D308, (PMID: 30418626).

See Also

meto.scan(), ac.scan(), me.scan(), p.scan(), su.scan(), gl.scan(), sni.scan(), ni.scan(), ptm.scan(), reg.scan(), dis.scan()

Examples

```
## Not run: ub.scan('Q16695', db = 'PSP')
```

`uniprot.kegg` *Identifier Mapping From UniProt to KEGG*

Description

Mapping between UniProt and KEGG protein identifiers.

Usage

```
uniprot.kegg(id)
```

Arguments

`id`	the identifier to be converted.

Value

Returns a character string corresponding to the requested identifier.

Author(s)

Juan Carlos Aledo

See Also

id.mapping()

Examples

```
## Not run: uniprot.kegg('P01009')
```

`uniprot.pdb` *Identifier Mapping From UniProt to PDB*

Description

Mapping between UniProt and PDB protein identifiers.

Usage

```
uniprot.pdb(id)
```

Arguments

`id`	the identifier to be converted.

Value

Returns a character string corresponding to the requested identifier.

Author(s)

Juan Carlos Aledo

See Also

id.mapping()

Examples

```
## Not run: uniprot.pdb('P01009')
```

`uniprot2pdb` *Return the PDB and Chain IDs of the Provided UniProt Protein*

Description

Returns the PDB and chain IDs of the provided protein.

Usage

```
uniprot2pdb(up_id)
```

Arguments

`up_id`	the UniProt ID.

Value

The function returns a dataframe with info about the PDB related to the protein of interest.

Author(s)

Juan Carlos Aledo

See Also

pdb2uniprot(), id.mapping()

Examples

```
## Not run: uniprot2pdb("P04406")
```

xprod *Compute Cross Product*

Description

Computes the cross product of two vectors in three-dimensional euclidean space.

Usage

```
xprod(...)
```

Arguments

`...`	vectors involved in the cross product.

Details

For each methionyl residue taking place in a S-aromatic motif, this function computes the aromatic residue involved, the distance between the delta sulfur and the aromatic ring's centroid, as well as the angle between the sulfur-aromatic vector and the normal vector of the plane containing the aromatic ring.

Value

This function returns a vector that is orthogonal to the plane containing the two vector used as arguments.

Author(s)

Juan Carlos Aledo

Examples

```
xprod(c(1,1,1), c(1,2,1))
```

Index