

Package ‘Rcpi’

December 16, 2025

Type Package

Version 1.46.0

Title Molecular Informatics Toolkit for Compound-Protein Interaction
in Drug Discovery

Description A molecular informatics toolkit with an integration of
bioinformatics and chemoinformatics tools for drug discovery.

License Artistic-2.0 | file LICENSE

URL <https://nanx.me/Rcpi/>, <https://github.com/nanxstats/Rcpi>

BugReports <https://github.com/nanxstats/Rcpi/issues>

Encoding UTF-8

LazyData true

VignetteBuilder knitr

Depends R (>= 3.0.2)

Imports Biostrings, GOSemSim, curl, doParallel, foreach, httr2,
jsonlite, methods, rlang, stats, utils

Suggests knitr, rmarkdown, testthat (>= 3.0.0)

biocViews Software, DataImport, DataRepresentation, FeatureExtraction,
Cheminformatics, BiomedicalInformatics, Proteomics, GO,
SystemsBiology

Config/testthat/edition 3

RoxygenNote 7.3.2

git_url <https://git.bioconductor.org/packages/Rcpi>

git_branch RELEASE_3_22

git_last_commit c114c2a

git_last_commit_date 2025-10-29

Repository Bioconductor 3.22

Date/Publication 2025-12-15

Author Nan Xiao [aut, cre] (ORCID: <<https://orcid.org/0000-0002-0250-5673>>),
Dong-Sheng Cao [aut],
Qing-Song Xu [aut]

Maintainer Nan Xiao <me@nanx.me>

Contents

Rcpi-package	5
AA2DACOR	6
AA3DMoRSE	6
AAACF	7
AABLOSUM100	7
AABLOSUM45	8
AABLOSUM50	8
AABLOSUM62	9
AABLOSUM80	9
AABurden	10
AAConn	10
AAConst	11
AACPSA	11
AADescAll	12
AAEdgeAdj	12
AAEigIdx	13
AAFGC	13
AGeom	14
AGETAWAY	14
AAindex	15
AAInfo	15
AAMetaInfo	16
AMOE2D	16
AMOE3D	17
AMolProp	17
APAM120	18
APAM250	18
APAM30	19
APAM40	19
APAM70	20
ARandic	20
ARDF	21
ATopo	21
ATopoChg	22
AWalk	22
AWHIM	23
acc	23
calcDrugFPSim	24
calcDrugMCSSim	25
calcParProtGOSim	27
calcParProtSeqSim	28
calcTwoProtGOSim	29
calcTwoProtSeqSim	30
checkProt	31
convMolFormat	32
extractDrugAIO	37
extractDrugALOGP	38
extractDrugAminoAcidCount	39
extractDrugApol	39
extractDrugAromaticAtomsCount	40

extractDrugAromaticBondsCount	41
extractDrugAtomCount	42
extractDrugAutocorrelationCharge	42
extractDrugAutocorrelationMass	43
extractDrugAutocorrelationPolarizability	44
extractDrugBCUT	45
extractDrugBondCount	46
extractDrugBPol	47
extractDrugCarbonTypes	48
extractDrugChiChain	49
extractDrugChiCluster	50
extractDrugChiPath	51
extractDrugChiPathCluster	52
extractDrugCPSA	53
extractDrugDescOB	54
extractDrugECI	55
extractDrugEstate	56
extractDrugEstateComplete	57
extractDrugExtended	58
extractDrugExtendedComplete	59
extractDrugFMF	60
extractDrugFragmentComplexity	61
extractDrugGraph	62
extractDrugGraphComplete	63
extractDrugGravitationalIndex	64
extractDrugHBondAcceptorCount	65
extractDrugHBondDonorCount	66
extractDrugHybridization	66
extractDrugHybridizationComplete	67
extractDrugHybridizationRatio	68
extractDrugIPMolecularLearning	69
extractDrugKappaShapeIndices	70
extractDrugKierHallSmarts	71
extractDrugKR	73
extractDrugKRComplete	74
extractDrugLargestChain	75
extractDrugLargestPiSystem	76
extractDrugLengthOverBreadth	76
extractDrugLongestAliphaticChain	77
extractDrugMACCS	78
extractDrugMACCSComplete	79
extractDrugMannholdLogP	79
extractDrugMDE	80
extractDrugMomentOfInertia	81
extractDrugOBFP2	82
extractDrugOBFP3	83
extractDrugOBFP4	84
extractDrugOBMACCS	85
extractDrugPetitjeanNumber	86
extractDrugPetitjeanShapeIndex	87
extractDrugPubChem	88
extractDrugPubChemComplete	88

extractDrugRotatableBondsCount	89
extractDrugRuleOfFive	90
extractDrugShortestPath	91
extractDrugShortestPathComplete	92
extractDrugStandard	93
extractDrugStandardComplete	94
extractDrugTPSA	95
extractDrugVABC	96
extractDrugVAdjMa	96
extractDrugWeight	97
extractDrugWeightedPath	98
extractDrugWHIM	99
extractDrugWienerNumbers	100
extractDrugXLogP	101
extractDrugZagrebIndex	102
extractPCMBLOSUM	103
extractPCMDescScales	104
extractPCMFAScales	105
extractPCMDSScales	106
extractPCMPPropScales	107
extractPCMScales	108
extractProtAAC	109
extractProtAPAAC	110
extractProtCTDC	112
extractProtCTDD	113
extractProtCTDT	114
extractProtCTriad	115
extractProtDC	115
extractProtGeary	116
extractProtMoran	118
extractProtMoreauBroto	120
extractProtPAAC	121
extractProtPSSM	123
extractProtPSSMAcc	126
extractProtPSSMFeature	127
extractProtQSO	128
extractProtSOCN	129
extractProtTC	130
getCPI	131
getDrug	132
getFASTAFromKEGG	133
getFASTAFromUniProt	133
getMolFromCAS	134
getMolFromChEMBL	135
getMolFromDrugBank	136
getMolFromKEGG	136
getMolFromPubChem	137
getPDBFromRCSBPDB	138
getPPI	139
getProt	140
getSeqFromKEGG	141
getSeqFromRCSBPDB	141

getSeqFromUniProt	142
getSmiFromChEMBL	143
getSmiFromDrugBank	144
getSmiFromKEGG	144
getSmiFromPubChem	145
OptAA3d	146
readFASTA	146
readMolFromSDF	147
readMolFromSmi	148
readPDB	149
searchDrug	150
segProt	151
Index	152

Rcpi-package

Rcpi: Molecular Informatics Toolkit for Compound-Protein Interaction in Drug Discovery

Description

A molecular informatics toolkit with an integration of bioinformatics and chemoinformatics tools for drug discovery.

Author(s)

Maintainer: Nan Xiao <me@nanx.me> ([ORCID](#))

Authors:

- Dong-Sheng Cao
- Qing-Song Xu <qsxu@csu.edu.cn>

See Also

Useful links:

- <https://nanx.me/Rcpi/>
- <https://github.com/nanxstats/Rcpi>
- Report bugs at <https://github.com/nanxstats/Rcpi/issues>

AA2DACOR	<i>2D Autocorrelations Descriptors for 20 Amino Acids calculated by Dragon</i>
----------	--

Description

2D Autocorrelations Descriptors for 20 Amino Acids calculated by Dragon

Details

This dataset includes the 2D autocorrelations descriptors of the 20 amino acids calculated by Dragon (version 5.4) used for scales extraction in this package.

Value

AA2DACOR data

Examples

```
data(AA2DACOR)
```

AA3DMoRSE	<i>3D-MoRSE Descriptors for 20 Amino Acids calculated by Dragon</i>
-----------	---

Description

3D-MoRSE Descriptors for 20 Amino Acids calculated by Dragon

Details

This dataset includes the 3D-MoRSE descriptors of the 20 amino acids calculated by Dragon (version 5.4) used for scales extraction in this package.

Value

AA3DMoRSE data

Examples

```
data(AA3DMoRSE)
```

AAACF*Atom-Centred Fragments Descriptors for 20 Amino Acids calculated by Dragon*

Description

Atom-Centred Fragments Descriptors for 20 Amino Acids calculated by Dragon

Details

This dataset includes the atom-centred fragments descriptors of the 20 amino acids calculated by Dragon (version 5.4) used for scales extraction in this package.

Value

AAACF data

Examples

```
data(AAACF)
```

AABLOSUM100*BLOSUM100 Matrix for 20 Amino Acids*

Description

BLOSUM100 Matrix for 20 Amino Acids

Details

BLOSUM100 Matrix for the 20 amino acids. The matrix was extracted from the Biostrings package of Bioconductor.

Value

AABLOSUM100 data

Examples

```
data(AABLOSUM100)
```

AABLOSUM45

BLOSUM45 Matrix for 20 Amino Acids

Description

BLOSUM45 Matrix for 20 Amino Acids

Details

BLOSUM45 Matrix for the 20 amino acids. The matrix was extracted from the Biostrings package of Bioconductor.

Value

AABLOSUM45 data

Examples

```
data(AABLOSUM45)
```

AABLOSUM50

BLOSUM50 Matrix for 20 Amino Acids

Description

BLOSUM50 Matrix for 20 Amino Acids

Details

BLOSUM50 Matrix for the 20 amino acids. The matrix was extracted from the Biostrings package of Bioconductor.

Value

AABLOSUM50 data

Examples

```
data(AABLOSUM50)
```

AABLOSUM62*BLOSUM62 Matrix for 20 Amino Acids*

Description

BLOSUM62 Matrix for 20 Amino Acids

Details

BLOSUM62 Matrix for the 20 amino acids. The matrix was extracted from the Biostrings package of Bioconductor.

Value

AABLOSUM62 data

Examples

```
data(AABLOSUM62)
```

AABLOSUM80*BLOSUM80 Matrix for 20 Amino Acids*

Description

BLOSUM80 Matrix for 20 Amino Acids

Details

BLOSUM80 Matrix for the 20 amino acids. The matrix was extracted from the Biostrings package of Bioconductor.

Value

AABLOSUM80 data

Examples

```
data(AABLOSUM80)
```

AABurden	<i>Burden Eigenvalues Descriptors for 20 Amino Acids calculated by Dragon</i>
----------	---

Description

Burden Eigenvalues Descriptors for 20 Amino Acids calculated by Dragon

Details

This dataset includes the Burden eigenvalues descriptors of the 20 amino acids calculated by Dragon (version 5.4) used for scales extraction in this package.

Value

AABurden data

Examples

```
data(AABurden)
```

AAConn	<i>Connectivity Indices Descriptors for 20 Amino Acids calculated by Dragon</i>
--------	---

Description

Connectivity Indices Descriptors for 20 Amino Acids calculated by Dragon

Details

This dataset includes the connectivity indices descriptors of the 20 amino acids calculated by Dragon (version 5.4) used for scales extraction in this package.

Value

AAConn data

Examples

```
data(AAConn)
```

AAConst*Constitutional Descriptors for 20 Amino Acids calculated by Dragon*

Description

Constitutional Descriptors for 20 Amino Acids calculated by Dragon

Details

This dataset includes the constitutional descriptors of the 20 amino acids calculated by Dragon (version 5.4) used for scales extraction in this package.

Value

AAConst data

Examples

```
data(AAConst)
```

AACPSA*CPSA Descriptors for 20 Amino Acids calculated by Discovery Studio*

Description

CPSA Descriptors for 20 Amino Acids calculated by Discovery Studio

Details

This dataset includes the CPSA descriptors of the 20 amino acids calculated by Discovery Studio (version 2.5) used for scales extraction in this package. All amino acid molecules had also been optimized with MOE 2011.10 (semiempirical AM1) before calculating these CPSA descriptors. The SDF file containing the information of the optimized amino acid molecules is included in this package. See [OptAA3d](#) for more information.

Value

AACPSA data

Examples

```
data(AACPSA)
```

AADescAll*All 2D Descriptors for 20 Amino Acids calculated by Dragon*

Description

All 2D Descriptors for 20 Amino Acids calculated by Dragon

Details

This dataset includes all the 2D descriptors of the 20 amino acids calculated by Dragon (version 5.4) used for scales extraction in this package.

Value

AADescAll data

Examples

```
data(AADescAll)
```

AAEdgeAdj*Edge Adjacency Indices Descriptors for 20 Amino Acids calculated by Dragon*

Description

Edge Adjacency Indices Descriptors for 20 Amino Acids calculated by Dragon

Details

This dataset includes the edge adjacency indices descriptors of the 20 amino acids calculated by Dragon (version 5.4) used for scales extraction in this package.

Value

AAEdgeAdj data

Examples

```
data(AAEdgeAdj)
```

AAEigIdx	<i>Eigenvalue-Based Indices Descriptors for 20 Amino Acids calculated by Dragon</i>
----------	---

Description

Eigenvalue-Based Indices Descriptors for 20 Amino Acids calculated by Dragon

Details

This dataset includes the eigenvalue-based indices descriptors of the 20 amino acids calculated by Dragon (version 5.4) used for scales extraction in this package.

Value

AAEigIdx data

Examples

```
data(AAEigIdx)
```

AAFGC	<i>Functional Group Counts Descriptors for 20 Amino Acids calculated by Dragon</i>
-------	--

Description

Functional Group Counts Descriptors for 20 Amino Acids calculated by Dragon

Details

This dataset includes the functional group counts descriptors of the 20 amino acids calculated by Dragon (version 5.4) used for scales extraction in this package.

Value

AAFGC data

Examples

```
data(AAFGC)
```

AAGeom

Geometrical Descriptors for 20 Amino Acids calculated by Dragon

Description

Geometrical Descriptors for 20 Amino Acids calculated by Dragon

Details

This dataset includes the geometrical descriptors of the 20 amino acids calculated by Dragon (version 5.4) used for scales extraction in this package.

Value

AAGeom data

Examples

```
data(AAGeom)
```

AAGETAWAY

GETAWAY Descriptors for 20 Amino Acids calculated by Dragon

Description

GETAWAY Descriptors for 20 Amino Acids calculated by Dragon

Details

This dataset includes the GETAWAY descriptors of the 20 amino acids calculated by Dragon (version 5.4) used for scales extraction in this package.

Value

AAGETAWAY data

Examples

```
data(AAGETAWAY)
```

AAindex	<i>AAindex Data of 544 Physicochemical and Biological Properties for 20 Amino Acids</i>
---------	---

Description

AAindex Data of 544 Physicochemical and Biological Properties for 20 Amino Acids

Details

The data was extracted from the AAindex1 database ver 9.1 (<ftp://ftp.genome.jp/pub/db/community/aaindex/aaindex1>) as of November 2012 (Data Last Modified 2006-08-14).

With this data, users could investigate each property's accession number and other details. Visit <https://www.genome.jp/dbget/aaindex.html> for more information.

Value

AAindex data

Examples

```
data(AAindex)
```

AAInfo	<i>Information Indices Descriptors for 20 Amino Acids calculated by Dragon</i>
--------	--

Description

Information Indices Descriptors for 20 Amino Acids calculated by Dragon

Details

This dataset includes the information indices descriptors of the 20 amino acids calculated by Dragon (version 5.4) used for scales extraction in this package.

Value

AAInfo data

Examples

```
data(AAInfo)
```

AAMetaInfo*Meta Information for the 20 Amino Acids*

Description

Meta Information for the 20 Amino Acids

Details

This dataset includes the meta information of the 20 amino acids used for the 2D and 3D descriptor calculation in this package. Each column represents:

AAName Amino acid name

Short One-letter representation

Abbreviation Three-letter representation

mol SMILES representation

PUBCHEM_COMPOUND_CID PubChem CID for the amino acid

PUBCHEM_LINK PubChem link for the amino acid

Value

AAMetaInfo data

Examples

```
data(AAMetaInfo)
```

AAMOE2D*2D Descriptors for 20 Amino Acids calculated by MOE 2011.10*

Description

2D Descriptors for 20 Amino Acids calculated by MOE 2011.10

Details

This dataset includes the 2D descriptors of the 20 amino acids calculated by MOE 2011.10 used for scales extraction in this package.

Value

AAMOE2D data

Examples

```
data(AAMOE2D)
```

AAMOE3D*3D Descriptors for 20 Amino Acids calculated by MOE 2011.10*

Description

3D Descriptors for 20 Amino Acids calculated by MOE 2011.10

Details

This dataset includes the 3D descriptors of the 20 amino acids calculated by MOE 2011.10 used for scales extraction in this package. All amino acid molecules had also been optimized with MOE (semiempirical AM1) before calculating these 3D descriptors. The SDF file containing the information of the optimized amino acid molecules is included in this package. See [OptAA3d](#) for more information.

Value

AAMOE3D data

Examples

```
data(AAMOE3D)
```

AAMolProp*Molecular Properties Descriptors for 20 Amino Acids calculated by Dragon*

Description

Molecular Properties Descriptors for 20 Amino Acids calculated by Dragon

Details

This dataset includes the molecular properties descriptors of the 20 amino acids calculated by Dragon (version 5.4) used for scales extraction in this package.

Value

AAMolProp data

Examples

```
data(AAMolProp)
```

AAPAM120

PAM120 Matrix for 20 Amino Acids

Description

PAM120 Matrix for 20 Amino Acids

Details

PAM120 Matrix for the 20 amino acids. The matrix was extracted from the Biostrings package of Bioconductor.

Value

AAPAM120 data

Examples

```
data(AAPAM120)
```

AAPAM250

PAM250 Matrix for 20 Amino Acids

Description

PAM250 Matrix for 20 Amino Acids

Details

PAM250 Matrix for the 20 amino acids. The matrix was extracted from the Biostrings package of Bioconductor.

Value

AAPAM250 data

Examples

```
data(AAPAM250)
```

AAPAM30

PAM30 Matrix for 20 Amino Acids

Description

PAM30 Matrix for 20 Amino Acids

Details

PAM30 Matrix for the 20 amino acids. The matrix was extracted from the Biostrings package of Bioconductor.

Value

AAPAM30 data

Examples

```
data(AAPAM30)
```

AAPAM40

PAM40 Matrix for 20 Amino Acids

Description

PAM40 Matrix for 20 Amino Acids

Details

PAM40 Matrix for the 20 amino acids. The matrix was extracted from the Biostrings package of Bioconductor.

Value

AAPAM40 data

Examples

```
data(AAPAM40)
```

AAPAM70

PAM70 Matrix for 20 Amino Acids

Description

PAM70 Matrix for 20 Amino Acids

Details

PAM70 Matrix for the 20 amino acids. The matrix was extracted from the Biostrings package of Bioconductor.

Value

AAPAM70 data

Examples

```
data(AAPAM70)
```

AARandic

Randic Molecular Profiles Descriptors for 20 Amino Acids calculated by Dragon

Description

Randic Molecular Profiles Descriptors for 20 Amino Acids calculated by Dragon

Details

This dataset includes the Randic molecular profiles descriptors of the 20 amino acids calculated by Dragon (version 5.4) used for scales extraction in this package.

Value

AARandic data

Examples

```
data(AARandic)
```

AARDF*RDF Descriptors for 20 Amino Acids calculated by Dragon*

Description

RDF Descriptors for 20 Amino Acids calculated by Dragon

Details

This dataset includes the RDF descriptors of the 20 amino acids calculated by Dragon (version 5.4) used for scales extraction in this package.

Value

AARDF data

Examples

```
data(AARDF)
```

AATopo*Topological Descriptors for 20 Amino Acids calculated by Dragon*

Description

Topological Descriptors for 20 Amino Acids calculated by Dragon

Details

This dataset includes the topological descriptors of the 20 amino acids calculated by Dragon (version 5.4) used for scales extraction in this package.

Value

AATopo data

Examples

```
data(AATopo)
```

AATopoChg	<i>Topological Charge Indices Descriptors for 20 Amino Acids calculated by Dragon</i>
-----------	---

Description

Topological Charge Indices Descriptors for 20 Amino Acids calculated by Dragon

Details

This dataset includes the topological charge indices descriptors of the 20 amino acids calculated by Dragon (version 5.4) used for scales extraction in this package.

Value

AATopoChg data

Examples

```
data(AATopoChg)
```

AAWalk	<i>Walk and Path Counts Descriptors for 20 Amino Acids calculated by Dragon</i>
--------	---

Description

Walk and Path Counts Descriptors for 20 Amino Acids calculated by Dragon

Details

This dataset includes the walk and path counts descriptors of the 20 amino acids calculated by Dragon (version 5.4) used for scales extraction in this package.

Value

AAWalk data

Examples

```
data(AAWalk)
```

AAWHIM

*WHIM Descriptors for 20 Amino Acids calculated by Dragon***Description**

WHIM Descriptors for 20 Amino Acids calculated by Dragon

Details

This dataset includes the WHIM descriptors of the 20 amino acids calculated by Dragon (version 5.4) used for scales extraction in this package.

Value

AAWHIM data

Examples

```
data(AAWHIM)
```

acc

*Auto Cross Covariance (ACC) for Generating Scales-Based Descriptors of the Same Length***Description**

Calculates auto covariance and auto cross covariance for generating scale-based descriptors of the same length.

Usage

```
acc(mat, lag)
```

Arguments

mat	A $p \times n$ matrix. Each row represents one scale (total p scales), each column represents one amino acid position (total n amino acids).
lag	The lag parameter. Must be less than the amino acids.

Value

A length $\text{lag} \times p^2$ named vector, the element names are constructed by: the scales index (crossed scales index) and lag index.

Note

To see more details about auto cross covariance, check the references.

References

Wold, S., Jonsson, J., Sjöström, M., Sandberg, M., & Rännar, S. (1993). DNA and peptide sequences and chemical processes multivariately modelled by principal component analysis and partial least-squares projections to latent structures. *Analytica chimica acta*, 277(2), 239–253.

Sjöström, M., Rännar, S., & Wieslander, Å. (1995). Polypeptide sequence property relationships in *Escherichia coli* based on auto cross covariances. *Chemometrics and intelligent laboratory systems*, 29(2), 295–305.

See Also

See [extractPCMScales](#) for generalized scales-based descriptors. For more details, see [extractPCMDescScales](#) and [extractPCMPPropScales](#).

Examples

```
p = 8      # p is the scales number
n = 200    # n is the amino acid number
lag = 7    # lag parameter
mat = matrix(rnorm(p * n), nrow = p, ncol = n)
acc(mat, lag)
```

calcDrugFPSim	<i>Calculate Drug Molecule Similarity Derived by Molecular Fingerprints</i>
---------------	---

Description

Calculate Drug Molecule Similarity Derived by Molecular Fingerprints

Usage

```
calcDrugFPSim(
  fp1,
  fp2,
  fptype = c("compact", "complete"),
  metric = c("tanimoto", "euclidean", "cosine", "dice", "hamming")
)
```

Arguments

fp1	The first molecule's fingerprints, could be extracted by <code>extractDrugMACCS()</code> , <code>extractDrugMACCSComplete()</code> etc.
fp2	The second molecule's fingerprints.
fptype	The fingerprint type, must be one of "compact" or "complete".
metric	The similarity metric, one of "tanimoto", "euclidean", "cosine", "dice" and "hamming".

Details

This function calculate drug molecule fingerprints similarity. Define a as the features of object A, b is the features of object B, c is the number of common features to A and B:

- Tanimoto: aka Jaccard - $c/a + b + c$
- Euclidean: $\sqrt{(a + b)}$
- Dice: aka Sorensen, Czekanowski, Hodgkin-Richards - $c/0.5[(a + c) + (b + c)]$
- Cosine: aka Ochiai, Carbo - $c/\sqrt{(a + c)(b + c)}$
- Hamming: aka Manhattan, taxi-cab, city-block distance - $(a + b)$

Value

The numeric similarity value.

References

Gasteiger, Johann, and Thomas Engel, eds. Chemoinformatics. Wiley.com, 2006.

Examples

```
mols = readMolFromSDF(system.file('compseq/tyrphostin.sdf', package = 'Rcpi'))

fp1 = extractDrugEstate(mols[[1]])
fp2 = extractDrugEstate(mols[[2]])
calcDrugFPSim(fp1, fp2, fptype = 'compact', metric = 'tanimoto')
calcDrugFPSim(fp1, fp2, fptype = 'compact', metric = 'euclidean')
calcDrugFPSim(fp1, fp2, fptype = 'compact', metric = 'cosine')
calcDrugFPSim(fp1, fp2, fptype = 'compact', metric = 'dice')
calcDrugFPSim(fp1, fp2, fptype = 'compact', metric = 'hamming')

fp3 = extractDrugEstateComplete(mols[[1]])
fp4 = extractDrugEstateComplete(mols[[2]])
calcDrugFPSim(fp3, fp4, fptype = 'complete', metric = 'tanimoto')
calcDrugFPSim(fp3, fp4, fptype = 'complete', metric = 'euclidean')
calcDrugFPSim(fp3, fp4, fptype = 'complete', metric = 'cosine')
calcDrugFPSim(fp3, fp4, fptype = 'complete', metric = 'dice')
calcDrugFPSim(fp3, fp4, fptype = 'complete', metric = 'hamming')
```

calcDrugMCSSim

Calculate Drug Molecule Similarity Derived by Maximum Common Substructure Search

Description

Calculate Drug Molecule Similarity Derived by Maximum Common Substructure Search

Usage

```
calcDrugMCSSim(  
  mol1,  
  mol2,  
  type = c("smile", "sdf"),  
  plot = FALSE,  
  al = 0,  
  au = 0,  
  bl = 0,  
  bu = 0,  
  matching.mode = "static",  
  ...  
)
```

Arguments

mol1	The first molecule. R character string object containing the molecule. See examples.
mol2	The second molecule. R character string object containing the molecule. See examples.
type	The input molecule format, 'smile' or 'sdf'.
plot	Logical. Should we plot the two molecules and their maximum common substructure?
al	Lower bound for the number of atom mismatches. Default is 0.
au	Upper bound for the number of atom mismatches. Default is 0.
bl	Lower bound for the number of bond mismatches. Default is 0.
bu	Upper bound for the number of bond mismatches. Default is 0.
matching.mode	Three modes for bond matching are supported: 'static', 'aromatic', and 'ring'.
...	Other graphical parameters

Details

This function calculate drug molecule similarity derived by maximum common substructure search. The maximum common substructure search algorithm is provided by the *fmcsR* package.

Value

A list containing the detail MCS information and similarity values. The numeric similarity value includes Tanimoto coefficient and overlap coefficient.

References

Wang, Y., Backman, T. W., Horan, K., & Girke, T. (2013). *fmcsR*: mismatch tolerant maximum common substructure searching in R. *Bioinformatics*, 29(21), 2792–2794.

Examples

```
mol1 = 'CC(C)CCCCC(=O)NCC1=CC(=C(C=C1)O)OC'
mol2 = 'O=C(NCc1cc(OC)c(O)cc1)CCCC/C=C/C(C)C'
mol3 = readChar(system.file('compseq/DB00859.sdf', package = 'Rcpi'), nchars = 1e+6)
mol4 = readChar(system.file('compseq/DB00860.sdf', package = 'Rcpi'), nchars = 1e+6)
## Not run:
sim1 = calcDrugMCSSim(mol1, mol2, type = 'smile')
sim2 = calcDrugMCSSim(mol3, mol4, type = 'sdf', plot = TRUE)
print(sim1[[2]]) # Tanimoto Coefficient
print(sim2[[3]]) # Overlap Coefficient

## End(Not run)
```

calcParProtGOSim	<i>Protein Sequence Similarity Calculation based on Gene Ontology (GO) Similarity</i>
------------------	---

Description

Protein Sequence Similarity Calculation based on Gene Ontology (GO) Similarity

Usage

```
calcParProtGOSim(
  golist,
  type = c("go", "gene"),
  ont = c("MF", "BP", "CC"),
  organism = "human",
  measure = "Resnik",
  combine = "BMA"
)
```

Arguments

golist	A character vector, each component contains a character vector of GO terms or one Entrez Gene ID.
type	Input type of golist, 'go' for GO Terms, 'gene' for gene ID.
ont	Default is 'MF', could be one of 'MF', 'BP', or 'CC' subontologies.
organism	Default is 'human', could be one of 'anopheles', 'arabidopsis', 'bovine', 'canine', 'chicken', 'chimp', 'coelicolor', 'ecolik12', 'ecsakai', 'fly', 'human', 'malaria', 'mouse', 'pig', 'rat', 'rhesus', 'worm', 'xenopus', 'yeast' or 'zebrafish'.
measure	Default is 'Resnik', could be one of 'Resnik', 'Lin', 'Rel', 'Jiang' or 'Wang'.
combine	Default is 'BMA', could be one of 'max', 'average', 'rcmax' or 'BMA' for combining semantic similarity scores of multiple GO terms associated with protein.

Details

This function calculates protein sequence similarity based on Gene Ontology (GO) similarity.

Value

A n x n similarity matrix.

See Also

See [calcTwoProtGOSim](#) for calculating the GO semantic similarity between two groups of GO terms or two Entrez gene IDs. See [calcParProtSeqSim](#) for paralleled protein similarity calculation based on sequence alignment.

Examples

```
# By GO Terms
go1 = c('GO:0005215', 'GO:0005488', 'GO:0005515', 'GO:0005625', 'GO:0005802', 'GO:0005905') # AP4B1
go2 = c('GO:0005515', 'GO:0005634', 'GO:0005681', 'GO:0008380', 'GO:0031202') # BCAS2
go3 = c('GO:0003735', 'GO:0005622', 'GO:0005840', 'GO:0006412') # PDE4DIP
glist = list(go1, go2, go3)
calcParProtGOSim(glist, type = 'go', ont = 'CC', measure = 'Wang')

# By Entrez gene id
genelist = list(c('150', '151', '152', '1814', '1815', '1816'))
calcParProtGOSim(genelist, type = 'gene', ont = 'BP', measure = 'Wang')
```

calcParProtSeqSim	<i>Parallellized Protein Sequence Similarity Calculation based on Sequence Alignment</i>
-------------------	--

Description

Parallellized Protein Sequence Similarity Calculation based on Sequence Alignment

Usage

```
calcParProtSeqSim(protlist, cores = 2, type = "local", submat = "BLOSUM62")
```

Arguments

protlist	A length n list containing n protein sequences, each component of the list is a character string, storing one protein sequence. Unknown sequences should be represented as ''.
cores	Integer. The number of CPU cores to use for parallel execution, default is 2. Users could use the detectCores() function in the parallel package to see how many cores they could use.
type	Type of alignment, default is 'local', could be 'global' or 'local', where 'global' represents Needleman-Wunsch global alignment; 'local' represents Smith-Waterman local alignment.
submat	Substitution matrix, default is 'BLOSUM62', could be one of 'BLOSUM45', 'BLOSUM50', 'BLOSUM62', 'BLOSUM80', 'BLOSUM100', 'PAM30', 'PAM40', 'PAM70', 'PAM120', 'PAM250'.

Details

This function implemented the parallellized version for calculating protein sequence similarity based on sequence alignment.

Value

A n x n similarity matrix.

See Also

See `calcTwoProtSeqSim` for protein sequence alignment for two protein sequences. See [calcParProtGOSim](#) for protein similarity calculation based on Gene Ontology (GO) semantic similarity.

Examples

```
s1 = readFASTA(system.file('protseq/P00750.fasta', package = 'Rcpi'))[[1]]
s2 = readFASTA(system.file('protseq/P08218.fasta', package = 'Rcpi'))[[1]]
s3 = readFASTA(system.file('protseq/P10323.fasta', package = 'Rcpi'))[[1]]
s4 = readFASTA(system.file('protseq/P20160.fasta', package = 'Rcpi'))[[1]]
s5 = readFASTA(system.file('protseq/Q9NZP8.fasta', package = 'Rcpi'))[[1]]
plist = list(s1, s2, s3, s4, s5)

psimmat = calcParProtSeqSim(plist, cores = 2, type = 'local',
                             submat = 'BLOSUM62')

print(psimmat)
```

calcTwoProtGOSim	<i>Protein Similarity Calculation based on Gene Ontology (GO) Similarity</i>
------------------	--

Description

Protein Similarity Calculation based on Gene Ontology (GO) Similarity

Usage

```
calcTwoProtGOSim(
  id1,
  id2,
  type = c("go", "gene"),
  ont = c("MF", "BP", "CC"),
  organism = "human",
  measure = "Resnik",
  combine = "BMA"
)
```

Arguments

id1	A character vector. length > 1: each element is a GO term; length = 1: the Entrez Gene ID.
id2	A character vector. length > 1: each element is a GO term; length = 1: the Entrez Gene ID.

type	Input type of id1 and id2, 'go' for GO Terms, 'gene' for gene ID.
ont	Default is 'MF', could be one of 'MF', 'BP', or 'CC' subontologies.
organism	Default is 'human', could be one of 'anopheles', 'arabidopsis', 'bovine', 'canine', 'chicken', 'chimp', 'coelicolor', 'ecolik12', 'ecsakai', 'fly', 'human', 'malaria', 'mouse', 'pig', 'rat', 'rhesus', 'worm', 'xenopus', 'yeast' or 'zebrafish'.
measure	Default is 'Resnik', could be one of 'Resnik', 'Lin', 'Rel', 'Jiang' or 'Wang'.
combine	Default is 'BMA', could be one of 'max', 'average', 'rcmax' or 'BMA' for combining semantic similarity scores of multiple GO terms associated with protein.

Details

This function calculates the Gene Ontology (GO) similarity between two groups of GO terms or two Entrez gene IDs.

Value

A n x n matrix.

See Also

See [calcParProtGOSim](#) for protein similarity calculation based on Gene Ontology (GO) semantic similarity. See [calcParProtSeqSim](#) for paralleled protein similarity calculation based on sequence alignment.

Examples

```
# By GO terms
go1 = c("GO:0004022", "GO:0004024", "GO:0004023")
go2 = c("GO:0009055", "GO:0020037")
calcTwoProtGOSim(go1, go2, type = 'go', ont = 'MF', measure = 'Wang')

# By Entrez gene id
gene1 = '241'
gene2 = '251'
calcTwoProtGOSim(gene1, gene2, type = 'gene', ont = 'CC', measure = 'Lin')
```

calcTwoProtSeqSim

Protein Sequence Alignment for Two Protein Sequences

Description

Protein Sequence Alignment for Two Protein Sequences

Usage

```
calcTwoProtSeqSim(seq1, seq2, type = "local", submat = "BLOSUM62")
```

Arguments

seq1	A character string, containing one protein sequence.
seq2	A character string, containing another protein sequence.
type	Type of alignment, default is 'local', could be 'global' or 'local', where 'global' represents Needleman-Wunsch global alignment; 'local' represents Smith-Waterman local alignment.
submat	Substitution matrix, default is 'BLOSUM62', could be one of 'BLOSUM45', 'BLOSUM50', 'BLOSUM62', 'BLOSUM80', 'BLOSUM100', 'PAM30', 'PAM40', 'PAM70', 'PAM120', 'PAM250'.

Details

This function implements the sequence alignment between two protein sequences.

Value

An Biostrings object containing the scores and other alignment information.

See Also

See [calcParProtSeqSim](#) for paralleled pairwise protein similarity calculation based on sequence alignment. See [calcTwoProtGOSim](#) for calculating the GO semantic similarity between two groups of GO terms or two Entrez gene IDs.

Examples

```
s1 = readFASTA(system.file('protseq/P00750.fasta', package = 'Rcpi'))[[1]]
s2 = readFASTA(system.file('protseq/P10323.fasta', package = 'Rcpi'))[[1]]

seqalign = calcTwoProtSeqSim(s1, s2)
seqalign
slot(seqalign, "score")
```

checkProt	<i>Check if the protein sequence's amino acid types are the 20 default types</i>
-----------	--

Description

Check if the protein sequence's amino acid types are the 20 default types

Usage

```
checkProt(x)
```

Arguments

x	A character vector, as the input protein sequence.
---	--

Details

This function checks if the protein sequence's amino acid types are the 20 default types.

Value

Logical. TRUE if all of the amino acid types of the sequence are within the 20 default types.

Examples

```
x = readFASTA(system.file('protseq/P00750.fasta', package = 'Rcpi'))[[1]]
checkProt(x) # TRUE
checkProt(paste(x, 'Z', sep = '')) # FALSE
```

convMolFormat

Chemical File Formats Conversion

Description

Chemical File Formats Conversion

Usage

```
convMolFormat(infile, outfile, from, to)
```

Arguments

infile	A character string. Indicating the input file location.
outfile	A character string. Indicating the output file location.
from	The format of infile. A character string supported by OpenBabel. See the note section for the supported formats.
to	The desired format of outfile. A character string supported by OpenBabel. See the note section for the supported formats.

Details

This function converts between various chemical file formats via OpenBabel. The complete supported file format list could be found at <https://openbabel.org/docs/FileFormats/Overview.html>.

Value

NULL

Note

The supported formats include:

- abinit – ABINIT Output Format [Read-only]
- acr – ACR format [Read-only]
- adf – ADF cartesian input format [Write-only]
- adfout – ADF output format [Read-only]
- alc – Alchemy format
- arc – Accelrys/MSI Biosym/Insight II CAR format [Read-only]

- axsf – XCrySDen Structure Format [Read-only]
- bgf – MSI BGF format
- box – Dock 3.5 Box format
- bs – Ball and Stick format
- c3d1 – Chem3D Cartesian 1 format
- c3d2 – Chem3D Cartesian 2 format
- cac – CAChe MolStruct format [Write-only]
- cacrt – Cacao Cartesian format
- cache – CAChe MolStruct format [Write-only]
- cacint – Cacao Internal format [Write-only]
- can – Canonical SMILES format
- car – Accelrys/MSI Biosym/Insight II CAR format [Read-only]
- castep – CASTEP format [Read-only]
- ccc – CCC format [Read-only]
- cdx – ChemDraw binary format [Read-only]
- cdxml – ChemDraw CDXML format
- cht – Chemtool format [Write-only]
- cif – Crystallographic Information File
- ck – ChemKin format
- cml – Chemical Markup Language
- cmlr – CML Reaction format
- com – Gaussian 98/03 Input [Write-only]
- CONFIG – DL-POLY CONFIG
- CONTCAR – VASP format [Read-only]
- copy – Copy raw text [Write-only]
- crk2d – Chemical Resource Kit diagram(2D)
- crk3d – Chemical Resource Kit 3D format
- csr – Accelrys/MSI Quanta CSR format [Write-only]
- cssr – CSD CSSR format [Write-only]
- ct – ChemDraw Connection Table format
- cub – Gaussian cube format
- cube – Gaussian cube format
- dat – Generic Output file format [Read-only]
- dmol – DMol3 coordinates format
- dx – OpenDX cube format for APBS
- ent – Protein Data Bank format
- fa – FASTA format
- fasta – FASTA format
- fch – Gaussian formatted checkpoint file format [Read-only]
- fchk – Gaussian formatted checkpoint file format [Read-only]

- fck – Gaussian formatted checkpoint file format [Read-only]
- feat – Feature format
- fh – Fenske-Hall Z-Matrix format [Write-only]
- fhiaims – FHIaims XYZ format
- fix – SMILES FIX format [Write-only]
- fpt – Fingerprint format [Write-only]
- fract – Free Form Fractional format
- fs – Fastsearch format
- fsa – FASTA format
- g03 – Gaussian Output [Read-only]
- g09 – Gaussian Output [Read-only]
- g92 – Gaussian Output [Read-only]
- g94 – Gaussian Output [Read-only]
- g98 – Gaussian Output [Read-only]
- gal – Gaussian Output [Read-only]
- gam – GAMESS Output [Read-only]
- gamess – GAMESS Output [Read-only]
- gamin – GAMESS Input
- gamout – GAMESS Output [Read-only]
- gau – Gaussian 98/03 Input [Write-only]
- gjc – Gaussian 98/03 Input [Write-only]
- gjf – Gaussian 98/03 Input [Write-only]
- got – GULP format [Read-only]
- gpr – Ghemical format
- gr96 – GROMOS96 format [Write-only]
- gro – GRO format
- gukin – GAMESS-UK Input
- gukout – GAMESS-UK Output
- gzmat – Gaussian Z-Matrix Input
- hin – HyperChem HIN format
- HISTORY – DL-POLY HISTORY [Read-only]
- inchi – InChI format
- inchikey – InChIKey [Write-only]
- inp – GAMESS Input
- ins – ShelX format [Read-only]
- jin – Jaguar input format [Write-only]
- jout – Jaguar output format [Read-only]
- k – Compare molecules using InChI [Write-only]
- log – Generic Output file format [Read-only]
- mcdl – MCDL format

- mcif – Macromolecular Crystallographic Info
- mdl – MDL MOL format
- ml2 – Sybyl Mol2 format
- mmCIF – Macromolecular Crystallographic Info
- mmd – MacroModel format
- mmod – MacroModel format
- mna – Multilevel Neighborhoods of Atoms (MNA) [Write-only]
- mol – MDL MOL format
- mol2 – Sybyl Mol2 format
- mold – Molden format
- molden – Molden format
- molf – Molden format
- molreport – Open Babel molecule report [Write-only]
- moo – MOPAC Output format [Read-only]
- mop – MOPAC Cartesian format
- mopert – MOPAC Cartesian format
- mopin – MOPAC Internal
- mopout – MOPAC Output format [Read-only]
- mp – Molpro input format [Write-only]
- mpc – MOPAC Cartesian format
- mpd – MolPrint2D format [Write-only]
- mpo – Molpro output format [Read-only]
- mpqc – MPQC output format [Read-only]
- mpqcin – MPQC simplified input format [Write-only]
- mrv – Chemical Markup Language
- msi – Accelrys/MSI Cerius II MSI format [Read-only]
- msms – M.F. Sanner's MSMS input format [Write-only]
- nul – Outputs nothing [Write-only]
- nw – NWChem input format [Write-only]
- nwo – NWChem output format [Read-only]
- out – Generic Output file format [Read-only]
- outmol – DMol3 coordinates format
- output – Generic Output file format [Read-only]
- pc – PubChem format [Read-only]
- pcm – PCModel Format
- pdb – Protein Data Bank format
- pdbqt – AutoDock PDQBT format
- png – PNG 2D depiction
- POSCAR – VASP format [Read-only]
- pov – POV-Ray input format [Write-only]

- pqr – PQR format
- pqs – Parallel Quantum Solutions format
- prep – Amber Prep format [Read-only]
- pwscf – PWscf format [Read-only]
- qcin – Q-Chem input format [Write-only]
- qcout – Q-Chem output format [Read-only]
- report – Open Babel report format [Write-only]
- res – ShelX format [Read-only]
- rsmi – Reaction SMILES format
- rxn – MDL RXN format
- sd – MDL MOL format
- sdf – MDL MOL format
- smi – SMILES format
- smiles – SMILES format
- svg – SVG 2D depiction [Write-only]
- sy2 – Sybyl Mol2 format
- t41 – ADF TAPE41 format [Read-only]
- tdd – Thermo format
- text – Read and write raw text
- therm – Thermo format
- tmol – TurboMole Coordinate format
- txt – Title format
- txyz – Tinker XYZ format
- unixyz – UniChem XYZ format
- vmol – ViewMol format
- xed – XED format [Write-only]
- xml – General XML format [Read-only]
- xsf – XCrySDen Structure Format [Read-only]
- xyz – XYZ cartesian coordinates format
- yob – YASARA.org YOB format
- zin – ZINDO input format [Write-only]

Examples

```
sdf = system.file('sysdata/OptAA3d.sdf', package = 'Rcpi')
# SDF to SMILES
## Not run:
convMolFormat(infile = sdf, outfile = 'aa.smi',
              from = 'sdf', to = 'smiles')
## End(Not run)
# SMILES to MOPAC Cartesian format
## Not run:
convMolFormat(infile = 'aa.smi', outfile = 'aa.mop',
              from = 'smiles', to = 'mop')
## End(Not run)
```

extractDrugAIO*Calculate All Molecular Descriptors in RcpI at Once*

Description

Calculate All Molecular Descriptors in RcpI at Once

Usage

```
extractDrugAIO(molecules, silent = TRUE, warn = TRUE)
```

Arguments

molecules	Parsed molecule object.
silent	Logical. Whether the calculating process should be shown or not, default is TRUE.
warn	Logical. Whether the warning about some descriptors need the 3D coordinates should be shown or not after the calculation, default is TRUE.

Details

This function calculates all the molecular descriptors in the RcpI package at once.

Value

A data frame, each row represents one of the molecules, each column represents one descriptor. Currently, this function returns total 293 descriptors composed of 48 descriptor types.

Note

Note that we need 3-D coordinates of the molecules to calculate some of the descriptors, if not provided, these descriptors values will be NA.

Examples

```
# Load 20 small molecules that have 3D coordinates
sdf = system.file('sysdata/OptAA3d.sdf', package = 'RcpI')

mol = readMolFromSDF(sdf)
dat = extractDrugAIO(mol, warn = FALSE)
```

extractDrugALOGP	<i>Calculate Atom Additive logP and Molar Refractivity Values Descriptor</i>
------------------	--

Description

Calculate Atom Additive logP and Molar Refractivity Values Descriptor

Usage

```
extractDrugALOGP(molecules, silent = TRUE)
```

Arguments

molecules	Parsed molecule object.
silent	Logical. Whether the calculating process should be shown or not, default is TRUE.

Details

Calculates ALOGP (Ghose-Crippen LogKow) and the Ghose-Crippen molar refractivity as described by Ghose, A.K. and Crippen, G.M. Note the underlying code in CDK assumes that aromaticity has been detected before evaluating this descriptor. The code also expects that the molecule will have hydrogens explicitly set. For SD files, this is usually not a problem since hydrogens are explicit. But for the case of molecules obtained from SMILES, hydrogens must be made explicit.

Value

A data frame, each row represents one of the molecules, each column represents one feature. This function returns three columns named ALogP, ALogp2 and AMR.

References

Ghose, A.K. and Crippen, G.M. , Atomic physicochemical parameters for three-dimensional structure-directed quantitative structure-activity relationships. I. Partition coefficients as a measure of hydrophobicity, *Journal of Computational Chemistry*, 1986, 7:565-577.

Ghose, A.K. and Crippen, G.M. , Atomic physicochemical parameters for three-dimensional-structure-directed quantitative structure-activity relationships. 2. Modeling dispersive and hydrophobic interactions, *Journal of Chemical Information and Computer Science*, 1987, 27:21-35.

Examples

```
smi = system.file('vignettesdata/FDAMDD.smi', package = 'Rcpi')  
  
mol = readMolFromSmi(smi, type = 'mol')  
dat = extractDrugALOGP(mol)  
head(dat)
```

`extractDrugAminoAcidCount`*Calculate the Number of Amino Acids Descriptor*

Description

Calculate the Number of Amino Acids Descriptor

Usage

```
extractDrugAminoAcidCount(molecules, silent = TRUE)
```

Arguments

<code>molecules</code>	Parsed molecule object.
<code>silent</code>	Logical. Whether the calculating process should be shown or not, default is TRUE.

Details

Calculates the number of each amino acids (total 20 types) found in the molecules.

Value

A data frame, each row represents one of the molecules, each column represents one feature. This function returns 20 columns named nA, nR, nN, nD, nC, nF, nQ, nE, nG, nH, nI, nP, nL, nK, nM, nS, nT, nY, nV, nW.

Examples

```
smi = system.file('vignettesdata/FDAMDD.smi', package = 'Rcpi')

mol = readMolFromSmi(smi, type = 'mol')
dat = extractDrugAminoAcidCount(mol)
head(dat)
```

`extractDrugApol`*Calculate the Sum of the Atomic Polarizabilities Descriptor*

Description

Calculate the Sum of the Atomic Polarizabilities Descriptor

Usage

```
extractDrugApol(molecules, silent = TRUE)
```

Arguments

molecules	Parsed molecule object.
silent	Logical. Whether the calculating process should be shown or not, default is TRUE.

Details

Calculates the sum of the atomic polarizabilities (including implicit hydrogens) descriptor. Polarizabilities are taken from <https://bit.ly/3PvNbhe>.

Value

A data frame, each row represents one of the molecules, each column represents one feature. This function returns one column named apol.

Examples

```
smi = system.file('vignettesdata/FDAMDD.smi', package = 'Rcpi')  
  
mol = readMolFromSmi(smi, type = 'mol')  
dat = extractDrugApol(mol)  
head(dat)
```

extractDrugAromaticAtomsCount

Calculate the Number of Aromatic Atoms Descriptor

Description

Calculate the Number of Aromatic Atoms Descriptor

Usage

```
extractDrugAromaticAtomsCount(molecules, silent = TRUE)
```

Arguments

molecules	Parsed molecule object.
silent	Logical. Whether the calculating process should be shown or not, default is TRUE.

Details

Calculates the number of aromatic atoms of a molecule.

Value

A data frame, each row represents one of the molecules, each column represents one feature. This function returns one column named naAromAtom.

Examples

```
smi = system.file('vignettesdata/FDAMDD.smi', package = 'Rcpi')

mol = readMolFromSmi(smi, type = 'mol')
dat = extractDrugAromaticAtomsCount(mol)
head(dat)
```

extractDrugAromaticBondsCount

Calculate the Number of Aromatic Bonds Descriptor

Description

Calculate the Number of Aromatic Bonds Descriptor

Usage

```
extractDrugAromaticBondsCount(molecules, silent = TRUE)
```

Arguments

molecules	Parsed molecule object.
silent	Logical. Whether the calculating process should be shown or not, default is TRUE.

Details

Calculates the number of aromatic bonds of a molecule.

Value

A data frame, each row represents one of the molecules, each column represents one feature. This function returns one column named nAromBond.

Examples

```
smi = system.file('vignettesdata/FDAMDD.smi', package = 'Rcpi')

mol = readMolFromSmi(smi, type = 'mol')
dat = extractDrugAromaticBondsCount(mol)
head(dat)
```

extractDrugAtomCount	<i>Calculate the Number of Atom Descriptor</i>
----------------------	--

Description

Calculate the Number of Atom Descriptor

Usage

```
extractDrugAtomCount(molecules, silent = TRUE)
```

Arguments

molecules	Parsed molecule object.
silent	Logical. Whether the calculating process should be shown or not, default is TRUE.

Details

Calculates the number of atoms of a certain element type in a molecule. By default it returns the count of all atoms.

Value

A data frame, each row represents one of the molecules, each column represents one feature. This function returns one column named nAtom.

Examples

```
smi = system.file('vignettesdata/FDAMDD.smi', package = 'Rcpi')  
  
mol = readMolFromSmi(smi, type = 'mol')  
dat = extractDrugAtomCount(mol)  
head(dat)
```

extractDrugAutocorrelationCharge	<i>Calculate the Moreau-Broto Autocorrelation Descriptors using Partial Charges</i>
----------------------------------	---

Description

Calculate the Moreau-Broto Autocorrelation Descriptors using Partial Charges

Usage

```
extractDrugAutocorrelationCharge(molecules, silent = TRUE)
```

Arguments

molecules	Parsed molecule object.
silent	Logical. Whether the calculating process should be shown or not, default is TRUE.

Details

Calculates the ATS autocorrelation descriptor, where the weight equal to the charges.

Value

A data frame, each row represents one of the molecules, each column represents one feature. This function returns 5 columns named ATSc1, ATSc2, ATSc3, ATSc4, ATSc5.

Examples

```
smi = system.file('vignettesdata/FDAMDD.smi', package = 'Rcpi')  
  
mol = readMolFromSmi(smi, type = 'mol')  
dat = extractDrugAutocorrelationCharge(mol)  
head(dat)
```

extractDrugAutocorrelationMass

Calculate the Moreau-Broto Autocorrelation Descriptors using Atomic Weight

Description

Calculate the Moreau-Broto Autocorrelation Descriptors using Atomic Weight

Usage

```
extractDrugAutocorrelationMass(molecules, silent = TRUE)
```

Arguments

molecules	Parsed molecule object.
silent	Logical. Whether the calculating process should be shown or not, default is TRUE.

Details

Calculates the ATS autocorrelation descriptor, where the weight equal to the scaled atomic mass.

Value

A data frame, each row represents one of the molecules, each column represents one feature. This function returns 5 columns named ATSm1, ATSm2, ATSm3, ATSm4, ATSm5.

References

Moreau, Gilles, and Pierre Broto. The autocorrelation of a topological structure: a new molecular descriptor. *Nouv. J. Chim* 4 (1980): 359-360.

Examples

```
smi = system.file('vignettesdata/FDAMDD.smi', package = 'Rcpi')

mol = readMolFromSmi(smi, type = 'mol')
dat = extractDrugAutocorrelationMass(mol)
head(dat)
```

extractDrugAutocorrelationPolarizability

Calculate the Moreau-Broto Autocorrelation Descriptors using Polarizability

Description

Calculate the Moreau-Broto Autocorrelation Descriptors using Polarizability

Usage

```
extractDrugAutocorrelationPolarizability(molecules, silent = TRUE)
```

Arguments

molecules	Parsed molecule object.
silent	Logical. Whether the calculating process should be shown or not, default is TRUE.

Details

Calculates the ATS autocorrelation descriptor using polarizability.

Value

A data frame, each row represents one of the molecules, each column represents one feature. This function returns 5 columns named ATSp1, ATSp2, ATSp3, ATSp4, ATSp5.

Examples

```
smi = system.file('vignettesdata/FDAMDD.smi', package = 'Rcpi')

mol = readMolFromSmi(smi, type = 'mol')
dat = extractDrugAutocorrelationPolarizability(mol)
head(dat)
```

extractDrugBCUT	<i>BCUT – Eigenvalue Based Descriptor</i>
-----------------	---

Description

BCUT – Eigenvalue Based Descriptor

Usage

```
extractDrugBCUT(molecules, silent = TRUE)
```

Arguments

molecules	Parsed molecule object.
silent	Logical. Whether the calculating process should be shown or not, default is TRUE.

Details

Eigenvalue based descriptor noted for its utility in chemical diversity. Described by Pearlman et al. The descriptor is based on a weighted version of the Burden matrix which takes into account both the connectivity as well as atomic properties of a molecule. The weights are a variety of atom properties placed along the diagonal of the Burden matrix. Currently three weighting schemes are employed:

- Atomic Weight
- Partial Charge (Gasteiger Marsilli)
- Polarizability (Kang et al.)

Value

A data frame, each row represents one of the molecules, each column represents one feature. This function returns 6 columns:

- BCUTw-1l, BCUTw-2l ... - n high lowest atom weighted BCUTS
- BCUTw-1h, BCUTw-2h ... - n low highest atom weighted BCUTS
- BCUTc-1l, BCUTc-2l ... - n high lowest partial charge weighted BCUTS
- BCUTc-1h, BCUTc-2h ... - n low highest partial charge weighted BCUTS
- BCUTp-1l, BCUTp-2l ... - n high lowest polarizability weighted BCUTS
- BCUTp-1h, BCUTp-2h ... - n low highest polarizability weighted BCUTS

Note

By default, the descriptor will return the highest and lowest eigenvalues for the three classes of descriptor in a single ArrayList (in the order shown above). However it is also possible to supply a parameter list indicating how many of the highest and lowest eigenvalues (for each class of descriptor) are required. The descriptor works with the hydrogen depleted molecule.

A side effect of specifying the number of highest and lowest eigenvalues is that it is possible to get two copies of all the eigenvalues. That is, if a molecule has 5 heavy atoms, then specifying the 5

highest eigenvalues returns all of them, and specifying the 5 lowest eigenvalues returns all of them, resulting in two copies of all the eigenvalues.

Note that it is possible to specify an arbitrarily large number of eigenvalues to be returned. However if the number (i.e., nhigh or nlow) is larger than the number of heavy atoms, the remaining eigenvalues will be NaN.

Given the above description, if the aim is to get all the eigenvalues for a molecule, you should set nlow to 0 and specify the number of heavy atoms (or some large number) for nhigh (or vice versa).

References

Pearlman, R.S. and Smith, K.M., Metric Validation and the Receptor-Relevant Subspace Concept, J. Chem. Inf. Comput. Sci., 1999, 39:28-35.

Burden, F.R., Molecular identification number for substructure searches, J. Chem. Inf. Comput. Sci., 1989, 29:225-227.

Burden, F.R., Chemically Intuitive Molecular Index, Quant. Struct. -Act. Relat., 1997, 16:309-314

Kang, Y.K. and Jhon, M.S., Additivity of Atomic Static Polarizabilities and Dispersion Coefficients, Theoretica Chimica Acta, 1982, 61:41-48

Examples

```
smi = system.file('vignettesdata/FDAMDD.smi', package = 'Rcpi')

mol = readMolFromSmi(smi, type = 'mol')
dat = extractDrugBCUT(mol)
head(dat)
```

extractDrugBondCount	<i>Calculate the Descriptor Based on the Number of Bonds of a Certain Bond Order</i>
----------------------	--

Description

Calculate the Descriptor Based on the Number of Bonds of a Certain Bond Order

Usage

```
extractDrugBondCount(molecules, silent = TRUE)
```

Arguments

molecules	Parsed molecule object.
silent	Logical. Whether the calculating process should be shown or not, default is TRUE.

Details

Calculates the descriptor based on the number of bonds of a certain bond order.

Value

A data frame, each row represents one of the molecules, each column represents one feature. This function returns one column named nB.

Examples

```
smi = system.file('vignettesdata/FDAMDD.smi', package = 'Rcpi')

mol = readMolFromSmi(smi, type = 'mol')
dat = extractDrugBondCount(mol)
head(dat)
```

extractDrugBPol	<i>Calculate the Descriptor that Describes the Sum of the Absolute Value of the Difference between Atomic Polarizabilities of All Bonded Atoms in the Molecule</i>
-----------------	--

Description

Calculates the Descriptor that Describes the Sum of the Absolute Value of the Difference between Atomic Polarizabilities of All Bonded Atoms in the Molecule

Usage

```
extractDrugBPol(molecules, silent = TRUE)
```

Arguments

molecules	Parsed molecule object.
silent	Logical. Whether the calculating process should be shown or not, default is TRUE.

Details

This descriptor calculates the sum of the absolute value of the difference between atomic polarizabilities of all bonded atoms in the molecule (including implicit hydrogens) with polarizabilities taken from <https://bit.ly/3PvNbhe>. This descriptor assumes 2-centered bonds.

Value

A data frame, each row represents one of the molecules, each column represents one feature. This function returns one column named bpol.

Examples

```
smi = system.file('vignettesdata/FDAMDD.smi', package = 'Rcpi')

mol = readMolFromSmi(smi, type = 'mol')
dat = extractDrugBPol(mol)
head(dat)
```

`extractDrugCarbonTypes`*Topological Descriptor Characterizing the Carbon Connectivity in Terms of Hybridization*

Description

Topological Descriptor Characterizing the Carbon Connectivity in Terms of Hybridization

Usage

```
extractDrugCarbonTypes(molecules, silent = TRUE)
```

Arguments

<code>molecules</code>	Parsed molecule object.
<code>silent</code>	Logical. Whether the calculating process should be shown or not, default is TRUE.

Details

Calculates the carbon connectivity in terms of hybridization. The function calculates 9 descriptors in the following order:

- C1SP1 - triply bound carbon bound to one other carbon
- C2SP1 - triply bound carbon bound to two other carbons
- C1SP2 - doubly bound carbon bound to one other carbon
- C2SP2 - doubly bound carbon bound to two other carbons
- C3SP2 - doubly bound carbon bound to three other carbons
- C1SP3 - singly bound carbon bound to one other carbon
- C2SP3 - singly bound carbon bound to two other carbons
- C3SP3 - singly bound carbon bound to three other carbons
- C4SP3 - singly bound carbon bound to four other carbons

Value

A data frame, each row represents one of the molecules, each column represents one feature. This function returns 9 columns named C1SP1, C2SP1, C1SP2, C2SP2, C3SP2, C1SP3, C2SP3, C3SP3 and C4SP3.

Examples

```
smi = system.file('vignettesdata/FDAMDD.smi', package = 'Rcpi')

mol = readMolFromSmi(smi, type = 'mol')
dat = extractDrugCarbonTypes(mol)
head(dat)
```

extractDrugChiChain	<i>Calculate the Kier and Hall Chi Chain Indices of Orders 3, 4, 5, 6 and 7</i>
---------------------	---

Description

Calculate the Kier and Hall Chi Chain Indices of Orders 3, 4, 5, 6 and 7

Usage

```
extractDrugChiChain(molecules, silent = TRUE)
```

Arguments

molecules	Parsed molecule object.
silent	Logical. Whether the calculating process should be shown or not, default is TRUE.

Details

Evaluates chi chain descriptors. The code currently evaluates the simple and valence chi chain descriptors of orders 3, 4, 5, 6 and 7. It utilizes the graph isomorphism code of the CDK to find fragments matching SMILES strings representing the fragments corresponding to each type of chain.

Value

A data frame, each row represents one of the molecules, each column represents one feature. This function returns 10 columns, in the following order:

- SCH.3 - Simple chain, order 3
- SCH.4 - Simple chain, order 4
- SCH.5 - Simple chain, order 5
- SCH.6 - Simple chain, order 6
- SCH.7 - Simple chain, order 7
- VCH.3 - Valence chain, order 3
- VCH.4 - Valence chain, order 4
- VCH.5 - Valence chain, order 5
- VCH.6 - Valence chain, order 6
- VCH.7 - Valence chain, order 7

Note

These descriptors are calculated using graph isomorphism to identify the various fragments. As a result calculations may be slow. In addition, recent versions of Molconn-Z use simplified fragment definitions (i.e., rings without branches etc.) whereas these descriptors use the older more complex fragment definitions.

Examples

```
smi = system.file('vignettesdata/FDAMDD.smi', package = 'Rcpi')

mol = readMolFromSmi(smi, type = 'mol')
dat = extractDrugChiChain(mol)
head(dat)
```

`extractDrugChiCluster` *Evaluates the Kier and Hall Chi cluster indices of orders 3, 4, 5 and 6*

Description

Evaluates the Kier and Hall Chi cluster indices of orders 3, 4, 5 and 6

Usage

```
extractDrugChiCluster(molecules, silent = TRUE)
```

Arguments

<code>molecules</code>	Parsed molecule object.
<code>silent</code>	Logical. Whether the calculating process should be shown or not, default is TRUE.

Details

Evaluates chi cluster descriptors. It utilizes the graph isomorphism code of the CDK to find fragments matching SMILES strings representing the fragments corresponding to each type of chain.

Value

A data frame, each row represents one of the molecules, each column represents one feature. This function returns 8 columns, the order and names of the columns returned is:

- SC.3 - Simple cluster, order 3
- SC.4 - Simple cluster, order 4
- SC.5 - Simple cluster, order 5
- SC.6 - Simple cluster, order 6
- VC.3 - Valence cluster, order 3
- VC.4 - Valence cluster, order 4
- VC.5 - Valence cluster, order 5
- VC.6 - Valence cluster, order 6

Note

These descriptors are calculated using graph isomorphism to identify the various fragments. As a result calculations may be slow. In addition, recent versions of Molconn-Z use simplified fragment definitions (i.e., rings without branches etc.) whereas these descriptors use the older more complex fragment definitions.

Examples

```
smi = system.file('vignettesdata/FDAMDD.smi', package = 'Rcpi')

mol = readMolFromSmi(smi, type = 'mol')
dat = extractDrugChiCluster(mol)
head(dat)
```

extractDrugChiPath	<i>Calculate the Kier and Hall Chi Path Indices of Orders 0 to 7</i>
--------------------	--

Description

Calculate the Kier and Hall Chi Path Indices of Orders 0 to 7

Usage

```
extractDrugChiPath(molecules, silent = TRUE)
```

Arguments

molecules	Parsed molecule object.
silent	Logical. Whether the calculating process should be shown or not, default is TRUE.

Details

Evaluates chi path descriptors. This function utilizes the graph isomorphism code of the CDK to find fragments matching SMILES strings representing the fragments corresponding to each type of chain.

Value

A data frame, each row represents one of the molecules, each column represents one feature. This function returns 16 columns, The order and names of the columns returned is:

- SP.0, SP.1, ..., SP.7 - Simple path, orders 0 to 7
- VP.0, VP.1, ..., VP.7 - Valence path, orders 0 to 7

Note

These descriptors are calculated using graph isomorphism to identify the various fragments. As a result calculations may be slow. In addition, recent versions of Molconn-Z use simplified fragment definitions (i.e., rings without branches etc.) whereas these descriptors use the older more complex fragment definitions.

Examples

```
smi = system.file('vignettesdata/FDAMDD.smi', package = 'Rcpi')

mol = readMolFromSmi(smi, type = 'mol')
dat = extractDrugChiPath(mol)
head(dat)
```

`extractDrugChiPathCluster`*Calculate the Kier and Hall Chi Path Cluster Indices of Orders 4, 5 and 6*

Description

Calculate the Kier and Hall Chi Path Cluster Indices of Orders 4, 5 and 6

Usage

```
extractDrugChiPathCluster(molecules, silent = TRUE)
```

Arguments

<code>molecules</code>	Parsed molecule object.
<code>silent</code>	Logical. Whether the calculating process should be shown or not, default is TRUE.

Details

Evaluates chi path cluster descriptors. The code currently evaluates the simple and valence chi chain descriptors of orders 4, 5 and 6. It utilizes the graph isomorphism code of the CDK to find fragments matching SMILES strings representing the fragments corresponding to each type of chain.

Value

A data frame, each row represents one of the molecules, each column represents one feature. This function returns 6 columns named SPC.4, SPC.5, SPC.6, VPC.4, VPC.5, VPC.6:

- SPC.4 - Simple path cluster, order 4
- SPC.5 - Simple path cluster, order 5
- SPC.6 - Simple path cluster, order 6
- VPC.4 - Valence path cluster, order 4
- VPC.5 - Valence path cluster, order 5
- VPC.6 - Valence path cluster, order 6

Note

These descriptors are calculated using graph isomorphism to identify the various fragments. As a result calculations may be slow. In addition, recent versions of Molconn-Z use simplified fragment definitions (i.e., rings without branches etc.) whereas these descriptors use the older more complex fragment definitions.

Examples

```
smi = system.file('vignettedata/FDAMDD.smi', package = 'Rcpi')  
  
mol = readMolFromSmi(smi, type = 'mol')  
dat = extractDrugChiPathCluster(mol)  
head(dat)
```

extractDrugCPSA	<i>A Variety of Descriptors Combining Surface Area and Partial Charge Information</i>
-----------------	---

Description

A Variety of Descriptors Combining Surface Area and Partial Charge Information

Usage

```
extractDrugCPSA(molecules, silent = TRUE)
```

Arguments

molecules	Parsed molecule object.
silent	Logical. Whether the calculating process should be shown or not, default is TRUE.

Details

Calculates 29 Charged Partial Surface Area (CPSA) descriptors. The CPSA's were developed by Stanton et al.

Value

A data frame, each row represents one of the molecules, each column represents one feature. This function returns 29 columns:

- PPSA.1 - partial positive surface area – sum of surface area on positive parts of molecule
- PPSA.2 - partial positive surface area * total positive charge on the molecule
- PPSA.3 - charge weighted partial positive surface area
- PNSA.1 - partial negative surface area – sum of surface area on negative parts of molecule
- PNSA.2 - partial negative surface area * total negative charge on the molecule
- PNSA.3 - charge weighted partial negative surface area
- DPSA.1 - difference of PPSA.1 and PNSA.1
- DPSA.2 - difference of PPSA.2 and PNSA.2
- DPSA.3 - difference of PPSA.3 and PNSA.3
- FPSA.1 - PPSA.1 / total molecular surface area
- FPSA.2 - PPSA.2 / total molecular surface area
- FPSA.3 - PPSA.3 / total molecular surface area
- FNSA.1 - PNSA.1 / total molecular surface area
- FNSA.2 - PNSA.2 / total molecular surface area
- FNSA.3 - PNSA.3 / total molecular surface area
- WPSA.1 - PPSA.1 * total molecular surface area / 1000
- WPSA.2 - PPSA.2 * total molecular surface area / 1000
- WPSA.3 - PPSA.3 * total molecular surface area / 1000

- WNSA.1 - PNSA.1 * total molecular surface area /1000
- WNSA.2 - PNSA.2 * total molecular surface area / 1000
- WNSA.3 - PNSA.3 * total molecular surface area / 1000
- RPCG - relative positive charge – most positive charge / total positive charge
- RNCG - relative negative charge – most negative charge / total negative charge
- RPCS - relative positive charge surface area – most positive surface area * RPCG
- RNCS - relative negative charge surface area – most negative surface area * RNCG
- THSA - sum of solvent accessible surface areas of atoms with absolute value of partial charges less than 0.2
- TPSA - sum of solvent accessible surface areas of atoms with absolute value of partial charges greater than or equal 0.2
- RHSA - THSA / total molecular surface area
- RPSA - TPSA / total molecular surface area

References

Stanton, D.T. and Jurs, P.C. , Development and Use of Charged Partial Surface Area Structural Descriptors in Computer Assisted Quantitative Structure Property Relationship Studies, Analytical Chemistry, 1990, 62:2323.2329.

Examples

```
sdf = system.file('sysdata/OptAA3d.sdf', package = 'Rcpi')  
  
mol = readMolFromSDF(sdf)  
dat = extractDrugCPSA(mol)  
head(dat)
```

extractDrugDescOB

Calculate Molecular Descriptors Provided by OpenBabel

Description

Calculate Molecular Descriptors Provided by OpenBabel

Usage

```
extractDrugDescOB(molecules, type = c("smile", "sdf"))
```

Arguments

molecules	R character string object containing the molecules. See the example section for details.
type	'smile' or 'sdf'.

Details

This function calculates 14 types of the *numerical* molecular descriptors provided in OpenBabel.

Value

A data frame, each row represents one of the molecules, each column represents one descriptor. This function returns 14 columns named abonds, atoms, bonds, dbonds, HBA1, HBA2, HBD, logP, MR, MW, nF, sbonds, tbonds, TPSA:

- abonds - Number of aromatic bonds
- atoms - Number of atoms
- bonds - Number of bonds
- dbonds - Number of double bonds
- HBA1 - Number of Hydrogen Bond Acceptors 1
- HBA2 - Number of Hydrogen Bond Acceptors 2
- HBD - Number of Hydrogen Bond Donors
- logP - Octanol/Water Partition Coefficient
- MR - Molar Refractivity
- MW - Molecular Weight Filter
- nF - Number of Fluorine Atoms
- sbonds - Number of single bonds
- tbonds - Number of triple bonds
- TPSA - Topological Polar Surface Area

Examples

```
mol1 = 'CC(=O)NCCC1=CNc2c1cc(OC)cc2' # one molecule SMILE in a vector
mol2 = c('OCCc1c(C)[n+](=cs1)Cc2cnc(C)nc(N)2',
        'CCc(c1)ccc2[n+]1ccc3c2Nc4c3cccc4',
        '[Cu+2].[O-]S(=O)(=O)[O-]') # multiple SMILES in a vector
mol3 = readChar(system.file('compseq/DB00860.sdf', package = 'Rcpi'),
                nchars = 1e+6) # single molecule in a sdf file
mol4 = readChar(system.file('sysdata/OptAA3d.sdf', package = 'Rcpi'),
                nchars = 1e+6) # multiple molecules in a sdf file

## Not run:
smidesc0 = extractDrugDescOB(mol1, type = 'smile')
smidesc1 = extractDrugDescOB(mol2, type = 'smile')
sdfdesc0 = extractDrugDescOB(mol3, type = 'sdf')
sdfdesc1 = extractDrugDescOB(mol4, type = 'sdf')
## End(Not run)
```

 extractDrugECI

Calculate the Eccentric Connectivity Index Descriptor

Description

Calculate the Eccentric Connectivity Index Descriptor

Usage

```
extractDrugECI(molecules, silent = TRUE)
```

Arguments

molecules	Parsed molecule object.
silent	Logical. Whether the calculating process should be shown or not, default is TRUE.

Details

Eccentric Connectivity Index (ECI) is a topological descriptor combining distance and adjacency information. This descriptor is described by Sharma et al. and has been shown to correlate well with a number of physical properties. The descriptor is also reported to have good discriminatory ability. The eccentric connectivity index for a hydrogen suppressed molecular graph is given by

$$x_i^c = \sum_{i=1}^n E(i)V(i)$$

where E(i) is the eccentricity of the i-th atom (path length from the i-th atom to the atom farthest from it) and V(i) is the vertex degree of the i-th atom.

Value

A data frame, each row represents one of the molecules, each column represents one feature. This function returns one column named ECCEN.

References

Sharma, V. and Goswami, R. and Madan, A.K. (1997), Eccentric Connectivity Index: A Novel Highly Discriminating Topological Descriptor for Structure-Property and Structure-Activity Studies, Journal of Chemical Information and Computer Sciences, 37:273-282

Examples

```
smi = system.file('vignettedata/FDAMDD.smi', package = 'Rcpi')

mol = readMolFromSmi(smi, type = 'mol')
dat = extractDrugECI(mol)
head(dat)
```

extractDrugEstate	<i>Calculate the E-State Molecular Fingerprints (in Compact Format)</i>
-------------------	---

Description

Calculate the E-State Molecular Fingerprints (in Compact Format)

Usage

```
extractDrugEstate(molecules, silent = TRUE)
```

Arguments

molecules	Parsed molecule object.
silent	Logical. Whether the calculating process should be shown or not, default is TRUE.

Details

79 bit fingerprints corresponding to the E-State atom types described by Hall and Kier.

Value

A list, each component represents one of the molecules, each element in the component represents the index of which element in the fingerprint is 1. Each component's name is the length of the fingerprints.

See Also

[extractDrugEstateComplete](#)

Examples

```
smi = system.file('vignettesdata/FDAMDD.smi', package = 'Rcpi')  
  
mol = readMolFromSmi(smi, type = 'mol')  
fp = extractDrugEstate(mol)  
head(fp)
```

extractDrugEstateComplete

Calculate the E-State Molecular Fingerprints (in Complete Format)

Description

Calculate the E-State Molecular Fingerprints (in Complete Format)

Usage

```
extractDrugEstateComplete(molecules, silent = TRUE)
```

Arguments

molecules	Parsed molecule object.
silent	Logical. Whether the calculating process should be shown or not, default is TRUE.

Details

79 bit fingerprints corresponding to the E-State atom types described by Hall and Kier.

Value

An integer vector or a matrix. Each row represents one molecule, the columns represent the fingerprints.

See Also

[extractDrugEstate](#)

Examples

```
smi = system.file('vignettesdata/FDAMDD.smi', package = 'Rcpi')

mol = readMolFromSmi(smi, type = 'mol')
fp = extractDrugEstateComplete(mol)
dim(fp)
```

extractDrugExtended	<i>Calculate the Extended Molecular Fingerprints (in Compact Format)</i>
---------------------	--

Description

Calculate the Extended Molecular Fingerprints (in Compact Format)

Usage

```
extractDrugExtended(molecules, depth = 6, size = 1024, silent = TRUE)
```

Arguments

molecules	Parsed molecule object.
depth	The search depth. Default is 6.
size	The length of the fingerprint bit string. Default is 1024.
silent	Logical. Whether the calculating process should be shown or not, default is TRUE.

Details

Calculate the extended molecular fingerprints. Considers paths of a given length, similar to the standard type, but takes rings and atomic properties into account. This is hashed fingerprints, with a default length of 1024.

Value

A list, each component represents one of the molecules, each element in the component represents the index of which element in the fingerprint is 1. Each component's name is the length of the fingerprints.

See Also

[extractDrugExtendedComplete](#)

Examples

```
smi = system.file('vignettesdata/FDAMDD.smi', package = 'Rcpi')

mol = readMolFromSmi(smi, type = 'mol')
fp = extractDrugExtended(mol)
head(fp)
```

`extractDrugExtendedComplete`*Calculate the Extended Molecular Fingerprints (in Complete Format)*

Description

Calculate the Extended Molecular Fingerprints (in Complete Format)

Usage

```
extractDrugExtendedComplete(molecules, depth = 6, size = 1024, silent = TRUE)
```

Arguments

<code>molecules</code>	Parsed molecule object.
<code>depth</code>	The search depth. Default is 6.
<code>size</code>	The length of the fingerprint bit string. Default is 1024.
<code>silent</code>	Logical. Whether the calculating process should be shown or not, default is TRUE.

Details

Calculate the extended molecular fingerprints. Considers paths of a given length, similar to the standard type, but takes rings and atomic properties into account. This is hashed fingerprints, with a default length of 1024.

Value

An integer vector or a matrix. Each row represents one molecule, the columns represent the fingerprints.

See Also

[extractDrugExtended](#)

Examples

```
smi = system.file('vignettesdata/FDAMDD.smi', package = 'Rcpi')  
  
mol = readMolFromSmi(smi, type = 'mol')  
fp = extractDrugExtendedComplete(mol)  
dim(fp)
```

`extractDrugFMF`*Calculate the FMF Descriptor*

Description

Calculate the FMF Descriptor

Usage

```
extractDrugFMF(molecules, silent = TRUE)
```

Arguments

<code>molecules</code>	Parsed molecule object.
<code>silent</code>	Logical. Whether the calculating process should be shown or not, default is TRUE.

Details

Calculates the FMF descriptor characterizing molecular complexity in terms of its Murcko framework. This descriptor is the ratio of heavy atoms in the framework to the total number of heavy atoms in the molecule. By definition, acyclic molecules which have no frameworks, will have a value of 0. Note that the authors consider an isolated ring system to be a framework (even though there is no linker).

Value

A data frame, each row represents one of the molecules, each column represents one feature. This function returns one column named FMF.

References

Yang, Y., Chen, H., Nilsson, I., Muresan, S., & Engkvist, O. (2010). Investigation of the relationship between topology and selectivity for druglike molecules. *Journal of medicinal chemistry*, 53(21), 7709-7714.

Examples

```
smi = system.file('vignettesdata/FDAMDD.smi', package = 'Rcpi')

mol = readMolFromSmi(smi, type = 'mol')
dat = extractDrugFMF(mol)
head(dat)
```

`extractDrugFragmentComplexity`*Calculate Complexity of a System*

Description

Calculate Complexity of a System

Usage

```
extractDrugFragmentComplexity(molecules, silent = TRUE)
```

Arguments

<code>molecules</code>	Parsed molecule object.
<code>silent</code>	Logical. Whether the calculating process should be shown or not, default is TRUE.

Details

This descriptor calculates the complexity of a system. The complexity is defined in Nilakantan, R. et al. as:

$$C = abs(B^2 - A^2 + A) + \frac{H}{100}$$

where C is complexity, A is the number of non-hydrogen atoms, B is the number of bonds and H is the number of heteroatoms.

Value

A data frame, each row represents one of the molecules, each column represents one feature. This function returns one column named fragC.

References

Nilakantan, R. and Nunn, D.S. and Greenblatt, L. and Walker, G. and Haraki, K. and Mobilio, D., A family of ring system-based structural fragments for use in structure-activity studies: database mining and recursive partitioning., Journal of chemical information and modeling, 2006, 46:1069-1077

Examples

```
smi = system.file('vignettesdata/FDAMDD.smi', package = 'Rcpi')  
  
mol = readMolFromSmi(smi, type = 'mol')  
dat = extractDrugFragmentComplexity(mol)  
head(dat)
```

extractDrugGraph	<i>Calculate the Graph Molecular Fingerprints (in Compact Format)</i>
------------------	---

Description

Calculate the Graph Molecular Fingerprints (in Compact Format)

Usage

```
extractDrugGraph(molecules, depth = 6, size = 1024, silent = TRUE)
```

Arguments

molecules	Parsed molecule object.
depth	The search depth. Default is 6.
size	The length of the fingerprint bit string. Default is 1024.
silent	Logical. Whether the calculating process should be shown or not, default is TRUE.

Details

Calculate the graph molecular fingerprints. Similar to the standard type by simply considers connectivity. This is hashed fingerprints, with a default length of 1024.

Value

A list, each component represents one of the molecules, each element in the component represents the index of which element in the fingerprint is 1. Each component's name is the length of the fingerprints.

See Also

[extractDrugGraphComplete](#)

Examples

```
smi = system.file('vignettesdata/FDAMDD.smi', package = 'Rcpi')  
  
mol = readMolFromSmi(smi, type = 'mol')  
fp = extractDrugGraph(mol)  
head(fp)
```

`extractDrugGraphComplete`*Calculate the Graph Molecular Fingerprints (in Complete Format)*

Description

Calculate the Graph Molecular Fingerprints (in Complete Format)

Usage

```
extractDrugGraphComplete(molecules, depth = 6, size = 1024, silent = TRUE)
```

Arguments

<code>molecules</code>	Parsed molecule object.
<code>depth</code>	The search depth. Default is 6.
<code>size</code>	The length of the fingerprint bit string. Default is 1024.
<code>silent</code>	Logical. Whether the calculating process should be shown or not, default is TRUE.

Details

Calculate the graph molecular fingerprints. Similar to the standard type by simply considers connectivity. This is hashed fingerprints, with a default length of 1024.

Value

An integer vector or a matrix. Each row represents one molecule, the columns represent the fingerprints.

See Also

[extractDrugGraph](#)

Examples

```
smi = system.file('vignettesdata/FDAMDD.smi', package = 'Rcpi')  
  
mol = readMolFromSmi(smi, type = 'mol')  
fp = extractDrugGraphComplete(mol)  
dim(fp)
```

`extractDrugGravitationalIndex`*Descriptor Characterizing the Mass Distribution of the Molecule.*

Description

Descriptor Characterizing the Mass Distribution of the Molecule.

Usage

```
extractDrugGravitationalIndex(molecules, silent = TRUE)
```

Arguments

<code>molecules</code>	Parsed molecule object.
<code>silent</code>	Logical. Whether the calculating process should be shown or not, default is TRUE.

Details

Descriptor characterizing the mass distribution of the molecule described by Katritzky et al. For modelling purposes the value of the descriptor is calculated both with and without H atoms. Furthermore the square and cube roots of the descriptor are also generated as described by Wessel et al.

Value

A data frame, each row represents one of the molecules, each column represents one feature. This function returns 9 columns:

- `GRAV.1` - gravitational index of heavy atoms
- `GRAV.2` - square root of gravitational index of heavy atoms
- `GRAV.3` - cube root of gravitational index of heavy atoms
- `GRAVH.1` - gravitational index - hydrogens included
- `GRAVH.2` - square root of hydrogen-included gravitational index
- `GRAVH.3` - cube root of hydrogen-included gravitational index
- `GRAV.4` - `grav1` for all pairs of atoms (not just bonded pairs)
- `GRAV.5` - `grav2` for all pairs of atoms (not just bonded pairs)
- `GRAV.6` - `grav3` for all pairs of atoms (not just bonded pairs)

References

Katritzky, A.R. and Mu, L. and Lobanov, V.S. and Karelson, M., Correlation of Boiling Points With Molecular Structure. 1. A Training Set of 298 Diverse Organics and a Test Set of 9 Simple Inorganics, J. Phys. Chem., 1996, 100:10400-10407.

Wessel, M.D. and Jurs, P.C. and Tolan, J.W. and Muskal, S.M. , Prediction of Human Intestinal Absorption of Drug Compounds From Molecular Structure, Journal of Chemical Information and Computer Sciences, 1998, 38:726-735.

Examples

```
sdf = system.file('sysdata/OptAA3d.sdf', package = 'Rcpi')

mol = readMolFromSDF(sdf)
dat = extractDrugGravitationalIndex(mol)
head(dat)
```

extractDrugHBondAcceptorCount

Number of Hydrogen Bond Acceptors

Description

Number of Hydrogen Bond Acceptors

Usage

```
extractDrugHBondAcceptorCount(molecules, silent = TRUE)
```

Arguments

molecules	Parsed molecule object.
silent	Logical. Whether the calculating process should be shown or not, default is TRUE.

Details

This descriptor calculates the number of hydrogen bond acceptors using a slightly simplified version of the PHACIR atom types. The following groups are counted as hydrogen bond acceptors: any oxygen where the formal charge of the oxygen is non-positive (i.e. formal charge ≤ 0) except

1. an aromatic ether oxygen (i.e. an ether oxygen that is adjacent to at least one aromatic carbon)
2. an oxygen that is adjacent to a nitrogen

and any nitrogen where the formal charge of the nitrogen is non-positive (i.e. formal charge ≤ 0) except a nitrogen that is adjacent to an oxygen.

Value

A data frame, each row represents one of the molecules, each column represents one feature. This function returns one column named nHBacc.

Examples

```
smi = system.file('vignettedata/FDAMDD.smi', package = 'Rcpi')

mol = readMolFromSmi(smi, type = 'mol')
dat = extractDrugHBondAcceptorCount(mol)
head(dat)
```

`extractDrugHBondDonorCount`*Number of Hydrogen Bond Donors*

Description

Number of Hydrogen Bond Donors

Usage

```
extractDrugHBondDonorCount(molecules, silent = TRUE)
```

Arguments

<code>molecules</code>	Parsed molecule object.
<code>silent</code>	Logical. Whether the calculating process should be shown or not, default is TRUE.

Details

This descriptor calculates the number of hydrogen bond donors using a slightly simplified version of the PHACIR atom types (<https://bit.ly/3qXQELf>). The following groups are counted as hydrogen bond donors:

- Any-OH where the formal charge of the oxygen is non-negative (i.e. formal charge ≥ 0)
- Any-NH where the formal charge of the nitrogen is non-negative (i.e. formal charge ≥ 0)

Value

A data frame, each row represents one of the molecules, each column represents one feature. This function returns one column named nHBDon.

Examples

```
smi = system.file('vignettesdata/FDAMDD.smi', package = 'Rcpi')  
  
mol = readMolFromSmi(smi, type = 'mol')  
dat = extractDrugHBondDonorCount(mol)  
head(dat)
```

`extractDrugHybridization`*Calculate the Hybridization Molecular Fingerprints (in Compact Format)*

Description

Calculate the Hybridization Molecular Fingerprints (in Compact Format)

Usage

```
extractDrugHybridization(molecules, depth = 6, size = 1024, silent = TRUE)
```

Arguments

molecules	Parsed molecule object.
depth	The search depth. Default is 6.
size	The length of the fingerprint bit string. Default is 1024.
silent	Logical. Whether the calculating process should be shown or not, default is TRUE.

Details

Calculate the hybridization molecular fingerprints. Similar to the standard type, but only consider hybridization state. This is hashed fingerprints, with a default length of 1024.

Value

A list, each component represents one of the molecules, each element in the component represents the index of which element in the fingerprint is 1. Each component's name is the length of the fingerprints.

See Also

[extractDrugHybridizationComplete](#)

Examples

```
smi = system.file('vignettedata/FDAMDD.smi', package = 'Rcpi')  
  
mol = readMolFromSmi(smi, type = 'mol')  
fp = extractDrugHybridization(mol)  
head(fp)
```

extractDrugHybridizationComplete

Calculate the Hybridization Molecular Fingerprints (in Complete Format)

Description

Calculate the Hybridization Molecular Fingerprints (in Complete Format)

Usage

```
extractDrugHybridizationComplete(  
  molecules,  
  depth = 6,  
  size = 1024,  
  silent = TRUE  
)
```

Arguments

molecules	Parsed molecule object.
depth	The search depth. Default is 6.
size	The length of the fingerprint bit string. Default is 1024.
silent	Logical. Whether the calculating process should be shown or not, default is TRUE.

Details

Calculate the hybridization molecular fingerprints. Similar to the standard type, but only consider hybridization state. This is hashed fingerprints, with a default length of 1024.

Value

An integer vector or a matrix. Each row represents one molecule, the columns represent the fingerprints.

See Also

[extractDrugHybridization](#)

Examples

```
smi = system.file('vignettesdata/FDAMDD.smi', package = 'Rcpi')

mol = readMolFromSmi(smi, type = 'mol')
fp = extractDrugHybridizationComplete(mol)
dim(fp)
```

extractDrugHybridizationRatio

Descriptor that Characterizing Molecular Complexity in Terms of Carbon Hybridization States

Description

Descriptor that Characterizing Molecular Complexity in Terms of Carbon Hybridization States

Usage

```
extractDrugHybridizationRatio(molecules, silent = TRUE)
```

Arguments

molecules	Parsed molecule object.
silent	Logical. Whether the calculating process should be shown or not, default is TRUE.

Details

This descriptor calculates the fraction of sp³ carbons to sp² carbons. Note that it only considers carbon atoms and rather than use a simple ratio it reports the value of $N_{sp3}/(N_{sp3} + N_{sp2})$. The original form of the descriptor (i.e., simple ratio) has been used to characterize molecular complexity, especially in the area of natural products, which usually have a high value of the sp³ to sp² ratio.

Value

A data frame, each row represents one of the molecules, each column represents one feature. This function returns one column named HybRatio.

Examples

```
smi = system.file('vignettesdata/FDAMDD.smi', package = 'Rcpi')  
  
mol = readMolFromSmi(smi, type = 'mol')  
dat = extractDrugHybridizationRatio(mol)  
head(dat)
```

extractDrugIPMolecularLearning

Calculate the Descriptor that Evaluates the Ionization Potential

Description

Calculate the Descriptor that Evaluates the Ionization Potential

Usage

```
extractDrugIPMolecularLearning(molecules, silent = TRUE)
```

Arguments

molecules	Parsed molecule object.
silent	Logical. Whether the calculating process should be shown or not, default is TRUE.

Details

Calculate the ionization potential of a molecule. The descriptor assumes that explicit hydrogens have been added to the molecules.

Value

A data frame, each row represents one of the molecules, each column represents one feature. This function returns one column named MolIP.

Examples

```
smi = system.file('vignettesdata/FDAMDD.smi', package = 'Rcpi')

mol = readMolFromSmi(smi, type = 'mol')
dat = extractDrugIPMolecularLearning(mol)
head(dat)
```

extractDrugKappaShapeIndices

Descriptor that Calculates Kier and Hall Kappa Molecular Shape Indices

Description

Descriptor that Calculates Kier and Hall Kappa Molecular Shape Indices

Usage

```
extractDrugKappaShapeIndices(molecules, silent = TRUE)
```

Arguments

molecules	Parsed molecule object.
silent	Logical. Whether the calculating process should be shown or not, default is TRUE.

Details

Kier and Hall Kappa molecular shape indices compare the molecular graph with minimal and maximal molecular graphs; see <https://bit.ly/3ramdBy> for details: "they are intended to capture different aspects of molecular shape. Note that hydrogens are ignored. In the following description, n denotes the number of atoms in the hydrogen suppressed graph, m is the number of bonds in the hydrogen suppressed graph. Also, let p2 denote the number of paths of length 2 and let p3 denote the number of paths of length 3".

Value

A data frame, each row represents one of the molecules, each column represents one feature. This function returns 3 columns named Kier1, Kier2 and Kier3:

- Kier1 - First kappa shape index
- Kier2 - Second kappa shape index
- Kier3 - Third kappa shape index

Examples

```
smi = system.file('vignettesdata/FDAMDD.smi', package = 'Rcpi')

mol = readMolFromSmi(smi, type = 'mol')
dat = extractDrugKappaShapeIndices(mol)
head(dat)
```

extractDrugKierHallSmarts

Descriptor that Counts the Number of Occurrences of the E-State Fragments

Description

Descriptor that Counts the Number of Occurrences of the E-State Fragments

Usage

```
extractDrugKierHallSmarts(molecules, silent = TRUE)
```

Arguments

molecules	Parsed molecule object.
silent	Logical. Whether the calculating process should be shown or not, default is TRUE.

Details

A fragment count descriptor that uses e-state fragments. Traditionally the e-state descriptors identify the relevant fragments and then evaluate the actual e-state value. However it has been shown in Butina et al. that simply using the counts of the e-state fragments can lead to QSAR models that exhibit similar performance to those built using the actual e-state indices.

Atom typing and aromaticity perception should be performed prior to calling this descriptor. The atom type definitions are taken from Hall et al. The SMARTS definitions were obtained from RDKit.

Value

A data frame, each row represents one of the molecules, each column represents one feature. This function returns 79 columns:

ID	Name	Pattern
0	khs.sLi	[LiD1]-*
1	khs.ssBe	[BeD2](-*)-*
2	khs.ssssBe	[BeD4](-*)(-*)(-*)-*
3	khs.ssBH	[BD2H](-*)-*
4	khs.sssB	[BD3](-*)(-*)-*
5	khs.ssssB	[BD4](-*)(-*)(-*)-*
6	khs.sCH3	[CD1H3]-*
7	khs.dCH2	[CD1H2]=*
8	khs.ssCH2	[CD2H2](-*)-*
9	khs.tCH	[CD1H]#*
10	khs.dsCH	[CD2H](=*)-*
11	khs.aaCH	[C,c;D2H](:*):*
12	khs.sssCH	[CD3H](-*)(-*)-*
13	khs.ddC	[CD2H0](=*)=*
14	khs.tsC	[CD2H0](#*)-*
15	khs.dssC	[CD3H0](=*)(-*)-*

16	khs.aasC	[C,c;D3H0](:*)(:*)-*
17	khs.aaaC	[C,c;D3H0](:*)(:*):*
18	khs.ssssC	[CD4H0](-*)(-*)(-*)-*
19	khs.sNH3	[ND1H3]-*
20	khs.sNH2	[ND1H2]-*
21	khs.ssNH2	[ND2H2](-*)-*
22	khs.dNH	[ND1H]=*
23	khs.ssNH	[ND2H](-*)-*
24	khs.aaNH	[N,nD2H](:*):*
25	khs.tN	[ND1H0]#*
26	khs.sssNH	[ND3H](-*)(-*)-*
27	khs.dsN	[ND2H0](=*)-*
28	khs.aaN	[N,nD2H0](:*):*
29	khs.sssN	[ND3H0](-*)(-*)-*
30	khs.ddsN	[ND3H0](~[OD1H0])(~[OD1H0])-,:*
31	khs.aasN	[N,nD3H0](:*)(:*)-,:*
32	khs.ssssN	[ND4H0](-*)(-*)(-*)-*
33	khs.sOH	[OD1H]-*
34	khs.dO	[OD1H0]=*
35	khs.ssO	[OD2H0](-*)-*
36	khs.aaO	[O,oD2H0](:*):*
37	khs.sF	[FD1]-*
38	khs.sSiH3	[SiD1H3]-*
39	khs.ssSiH2	[SiD2H2](-*)-*
40	khs.sssSiH	[SiD3H1](-*)(-*)-*
41	khs.ssssSi	[SiD4H0](-*)(-*)(-*)-*
42	khs.sPH2	[PD1H2]-*
43	khs.ssPH	[PD2H1](-*)-*
44	khs.sssP	[PD3H0](-*)(-*)-*
45	khs.dsssP	[PD4H0](=*)(-*)(-*)-*
46	khs.sssssP	[PD5H0](-*)(-*)(-*)(-*)-*
47	khs.sSH	[SD1H1]-*
48	khs.dS	[SD1H0]=*
49	khs.ssS	[SD2H0](-*)-*
50	khs.aaS	[S,sD2H0](:*):*
51	khs.dssS	[SD3H0](=*)(-*)-*
52	khs.ddssS	[SD4H0](~[OD1H0])(~[OD1H0])(-*)-*
53	khs.sCl	[ClD1]-*
54	khs.sGeH3	[GeD1H3](-*)
55	khs.ssGeH2	[GeD2H2](-*)-*
56	khs.sssGeH	[GeD3H1](-*)(-*)-*
57	khs.ssssGe	[GeD4H0](-*)(-*)(-*)-*
58	khs.sAsH2	[AsD1H2]-*
59	khs.ssAsH	[AsD2H1](-*)-*
60	khs.sssAs	[AsD3H0](-*)(-*)-*
61	khs.sssdAs	[AsD4H0](=*)(-*)(-*)-*
62	khs.sssssAs	[AsD5H0](-*)(-*)(-*)(-*)-*
63	khs.sSeH	[SeD1H1]-*
64	khs.dSe	[SeD1H0]=*
65	khs.ssSe	[SeD2H0](-*)-*
66	khs.aaSe	[SeD2H0](:*):*
67	khs.dssSe	[SeD3H0](=*)(-*)-*

68	khs.ddssSe	[SeD4H0](=*)(=*)(-*)-*
69	khs.sBr	[BrD1]-*
70	khs.sSnH3	[SnD1H3]-*
71	khs.ssSnH2	[SnD2H2](-*)-*
72	khs.sssSnH	[SnD3H1](-*)(-*)-*
73	khs.ssssSn	[SnD4H0](-*)(-*)(-*)-*
74	khs.sI	[ID1]-*
75	khs.sPbH3	[PbD1H3]-*
76	khs.ssPbH2	[PbD2H2](-*)-*
77	khs.sssPbH	[PbD3H1](-*)(-*)-*
78	khs.ssssPb	[PbD4H0](-*)(-*)(-*)-*

References

Butina, D. , Performance of Kier-Hall E-state Descriptors in Quantitative Structure Activity Relationship (QSAR) Studies of Multifunctional Molecules, *Molecules*, 2004, 9:1004-1009.

Hall, L.H. and Kier, L.B. , Electrotological State Indices for Atom Types: A Novel Combination of Electronic, Topological, and Valence State Information, *Journal of Chemical Information and Computer Science*, 1995, 35:1039-1045.

Examples

```
smi = system.file('vignettesdata/FDAMDD.smi', package = 'Rcpi')

mol = readMolFromSmi(smi, type = 'mol')
dat = extractDrugKierHallSmarts(mol)
head(dat)
```

extractDrugKR	<i>Calculate the KR (Klekota and Roth) Molecular Fingerprints (in Compact Format)</i>
---------------	---

Description

Calculate the KR (Klekota and Roth) Molecular Fingerprints (in Compact Format)

Usage

```
extractDrugKR(molecules, silent = TRUE)
```

Arguments

molecules	Parsed molecule object.
silent	Logical. Whether the calculating process should be shown or not, default is TRUE.

Details

Calculate the 4860 bit fingerprint defined by Klekota and Roth.

Value

A list, each component represents one of the molecules, each element in the component represents the index of which element in the fingerprint is 1. Each component's name is the length of the fingerprints.

See Also

[extractDrugKRComplete](#)

Examples

```
smi = system.file('vignettesdata/FDAMDD.smi', package = 'Rcpi')  
  
mol = readMolFromSmi(smi, type = 'mol')  
fp = extractDrugKR(mol)  
head(fp)
```

extractDrugKRComplete	<i>Calculate the KR (Klekota and Roth) Molecular Fingerprints (in Complete Format)</i>
-----------------------	--

Description

Calculate the KR (Klekota and Roth) Molecular Fingerprints (in Complete Format)

Usage

```
extractDrugKRComplete(molecules, silent = TRUE)
```

Arguments

molecules	Parsed molecule object.
silent	Logical. Whether the calculating process should be shown or not, default is TRUE.

Details

Calculate the 4860 bit fingerprint defined by Klekota and Roth.

Value

An integer vector or a matrix. Each row represents one molecule, the columns represent the fingerprints.

See Also

[extractDrugKR](#)

Examples

```
smi = system.file('vignettesdata/FDAMDD.smi', package = 'Rcpi')

mol = readMolFromSmi(smi, type = 'mol')
fp = extractDrugKRComplete(mol)
dim(fp)
```

extractDrugLargestChain

Descriptor that Calculates the Number of Atoms in the Largest Chain

Description

Descriptor that Calculates the Number of Atoms in the Largest Chain

Usage

```
extractDrugLargestChain(molecules, silent = TRUE)
```

Arguments

molecules	Parsed molecule object.
silent	Logical. Whether the calculating process should be shown or not, default is TRUE.

Details

This descriptor calculates the number of atoms in the largest chain. Note that a chain exists if there are two or more atoms. Thus single atom molecules will return 0.

Value

A data frame, each row represents one of the molecules, each column represents one feature. This function returns one column named nAtomLC.

Examples

```
smi = system.file('vignettesdata/FDAMDD.smi', package = 'Rcpi')

mol = readMolFromSmi(smi, type = 'mol')
dat = extractDrugLargestChain(mol)
head(dat)
```

`extractDrugLargestPiSystem`*Descriptor that Calculates the Number of Atoms in the Largest Pi Chain*

Description

Descriptor that Calculates the Number of Atoms in the Largest Pi Chain

Usage

```
extractDrugLargestPiSystem(molecules, silent = TRUE)
```

Arguments

<code>molecules</code>	Parsed molecule object.
<code>silent</code>	Logical. Whether the calculating process should be shown or not, default is TRUE.

Details

This descriptor calculates the number of atoms in the largest pi chain.

Value

A data frame, each row represents one of the molecules, each column represents one feature. This function returns one column named `nAtomP`.

Examples

```
smi = system.file('vignettesdata/FDAMDD.smi', package = 'Rcpi')  
  
mol = readMolFromSmi(smi, type = 'mol')  
dat = extractDrugLargestPiSystem(mol)  
head(dat)
```

`extractDrugLengthOverBreadth`*Calculate the Ratio of Length to Breadth Descriptor*

Description

Calculate the Ratio of Length to Breadth Descriptor

Usage

```
extractDrugLengthOverBreadth(molecules, silent = TRUE)
```

Arguments

molecules	Parsed molecule object.
silent	Logical. Whether the calculating process should be shown or not, default is TRUE.

Details

Calculates the Ratio of Length to Breadth, as a result it does not perform any orientation and only considers the X & Y extents for a series of rotations about the Z axis (in 10 degree increments).

Value

A data frame, each row represents one of the molecules, each column represents one feature. This function returns two columns named LOBMAX and LOBMIN:

- LOBMAX - The maximum L/B ratio;
- LOBMIN - The L/B ratio for the rotation that results in the minimum area (defined by the product of the X & Y extents for that orientation).

Note

The descriptor assumes that the atoms have been configured.

Examples

```
sdf = system.file('sysdata/OptAA3d.sdf', package = 'Rcpi')  
  
mol = readMolFromSDF(sdf)  
dat = extractDrugLengthOverBreadth(mol)  
head(dat)
```

extractDrugLongestAliphaticChain

Descriptor that Calculates the Number of Atoms in the Longest Aliphatic Chain

Description

Descriptor that Calculates the Number of Atoms in the Longest Aliphatic Chain

Usage

```
extractDrugLongestAliphaticChain(molecules, silent = TRUE)
```

Arguments

molecules	Parsed molecule object.
silent	Logical. Whether the calculating process should be shown or not, default is TRUE.

Details

This descriptor calculates the number of atoms in the longest aliphatic chain.

Value

A data frame, each row represents one of the molecules, each column represents one feature. This function returns one column named nAtomLAC.

Examples

```
smi = system.file('vignettedata/FDAMDD.smi', package = 'Rcpi')  
  
mol = readMolFromSmi(smi, type = 'mol')  
dat = extractDrugLongestAliphaticChain(mol)  
head(dat)
```

extractDrugMACCS	<i>Calculate the MACCS Molecular Fingerprints (in Compact Format)</i>
------------------	---

Description

Calculate the MACCS Molecular Fingerprints (in Compact Format)

Usage

```
extractDrugMACCS(molecules, silent = TRUE)
```

Arguments

molecules	Parsed molecule object.
silent	Logical. Whether the calculating process should be shown or not, default is TRUE.

Details

The popular 166 bit MACCS keys described by MDL.

Value

A list, each component represents one of the molecules, each element in the component represents the index of which element in the fingerprint is 1. Each component's name is the length of the fingerprints.

See Also

[extractDrugMACCSComplete](#)

Examples

```
smi = system.file('vignettedata/FDAMDD.smi', package = 'Rcpi')  
  
mol = readMolFromSmi(smi, type = 'mol')  
fp = extractDrugMACCS(mol)  
head(fp)
```

`extractDrugMACCSComplete`*Calculate the MACCS Molecular Fingerprints (in Complete Format)*

Description

Calculate the MACCS Molecular Fingerprints (in Complete Format)

Usage

```
extractDrugMACCSComplete(molecules, silent = TRUE)
```

Arguments

<code>molecules</code>	Parsed molecule object.
<code>silent</code>	Logical. Whether the calculating process should be shown or not, default is TRUE.

Details

The popular 166 bit MACCS keys described by MDL.

Value

An integer vector or a matrix. Each row represents one molecule, the columns represent the fingerprints.

See Also

[extractDrugMACCS](#)

Examples

```
smi = system.file('vignettedata/FDAMDD.smi', package = 'Rcpi')  
  
mol = readMolFromSmi(smi, type = 'mol')  
fp = extractDrugMACCSComplete(mol)  
dim(fp)
```

`extractDrugMannholdLogP`*Descriptor that Calculates the LogP Based on a Simple Equation Using the Number of Carbons and Hetero Atoms*

Description

Descriptor that Calculates the LogP Based on a Simple Equation Using the Number of Carbons and Hetero Atoms

Usage

```
extractDrugMannholdLogP(molecules, silent = TRUE)
```

Arguments

molecules	Parsed molecule object.
silent	Logical. Whether the calculating process should be shown or not, default is TRUE.

Details

This descriptor calculates the LogP based on a simple equation using the number of carbons and hetero atoms. The implemented equation was proposed in Mannhold et al.

Value

A data frame, each row represents one of the molecules, each column represents one feature. This function returns one column named MLogP.

References

Mannhold, R., Poda, G. I., Ostermann, C., & Tetko, I. V. (2009). Calculation of molecular lipophilicity: State-of-the-art and comparison of log P methods on more than 96,000 compounds. *Journal of pharmaceutical sciences*, 98(3), 861-893.

Examples

```
smi = system.file('vignettesdata/FDAMDD.smi', package = 'Rcpi')  
  
mol = readMolFromSmi(smi, type = 'mol')  
dat = extractDrugMannholdLogP(mol)  
head(dat)
```

extractDrugMDE	<i>Calculate Molecular Distance Edge (MDE) Descriptors for C, N and O</i>
----------------	---

Description

Calculate Molecular Distance Edge (MDE) Descriptors for C, N and O

Usage

```
extractDrugMDE(molecules, silent = TRUE)
```

Arguments

molecules	Parsed molecule object.
silent	Logical. Whether the calculating process should be shown or not, default is TRUE.

Details

This descriptor calculates the 10 molecular distance edge (MDE) descriptor described in Liu, S., Cao, C., & Li, Z., and in addition it calculates variants where O and N are considered.

Value

A data frame, each row represents one of the molecules, each column represents one feature. This function returns one column named nAtomLAC.

References

Liu, S., Cao, C., & Li, Z. (1998). Approach to estimation and prediction for normal boiling point (NBP) of alkanes based on a novel molecular distance-edge (MDE) vector, lambda. Journal of chemical information and computer sciences, 38(3), 387-394.

Examples

```
smi = system.file('vignettedata/FDAMDD.smi', package = 'Rcpi')

mol = readMolFromSmi(smi, type = 'mol')
dat = extractDrugMDE(mol)
head(dat)
```

extractDrugMomentOfInertia

Descriptor that Calculates the Principal Moments of Inertia and Ratios of the Principal Moments

Description

Descriptor that Calculates the Principal Moments of Inertia and Ratios of the Principal Moments

Usage

```
extractDrugMomentOfInertia(molecules, silent = TRUE)
```

Arguments

molecules	Parsed molecule object.
silent	Logical. Whether the calculating process should be shown or not, default is TRUE.

Details

A descriptor that calculates the moment of inertia and radius of gyration. Moment of inertia (MI) values characterize the mass distribution of a molecule. Related to the MI values, ratios of the MI values along the three principal axes are also well know modeling variables. This descriptor calculates the MI values along the X, Y and Z axes as well as the ratio's X/Y, X/Z and Y/Z. Finally it also calculates the radius of gyration of the molecule.

Value

A data frame, each row represents one of the molecules, each column represents one feature. This function returns 7 columns named MOMI.X, MOMI.Y, MOMI.Z, MOMI.XY, MOMI.XZ, MOMI.YZ, MOMI.R:

- MOMI.X - MI along X axis
- MOMI.Y - MI along Y axis
- MOMI.Z - MI along Z axis
- MOMI.XY - X/Y
- MOMI.XZ - X/Z
- MOMI.YZ - Y/Z
- MOMI.R - Radius of gyration

One important aspect of the algorithm is that if the eigenvalues of the MI tensor are below $1e-3$, then the ratio's are set to a default of 1000.

Examples

```
sdf = system.file('sysdata/OptAA3d.sdf', package = 'Rcpi')  
  
mol = readMolFromSDF(sdf)  
dat = extractDrugMomentOfInertia(mol)  
head(dat)
```

extractDrugOBFP2

Calculate the FP2 Molecular Fingerprints

Description

Calculate the FP2 Molecular Fingerprints

Usage

```
extractDrugOBFP2(molecules, type = c("smile", "sdf"))
```

Arguments

molecules	R character string object containing the molecules. See the example section for details.
type	'smile' or 'sdf'.

Details

Calculate the 1024 bit FP2 fingerprints provided by OpenBabel.

Value

A matrix. Each row represents one molecule, the columns represent the fingerprints.

Examples

```
mol1 = 'C1CCC1CC(CN(C)(C))CC(=O)CC' # one molecule SMILE in a vector
mol2 = c('CCC', 'CCN', 'CCN(C)(C)', 'c1ccccc1Cc1ccccc1',
        'C1CCC1CC(CN(C)(C))CC(=O)CC') # multiple SMILES in a vector
mol3 = readChar(system.file('compseq/DB00860.sdf', package = 'Rcpi'),
                nchars = 1e+6) # single molecule in a sdf file
mol4 = readChar(system.file('sysdata/OptAA3d.sdf', package = 'Rcpi'),
                nchars = 1e+6) # multiple molecules in a sdf file

## Not run:
smifp0 = extractDrugOBFP2(mol1, type = 'smile')
smifp1 = extractDrugOBFP2(mol2, type = 'smile')
sdffp0 = extractDrugOBFP2(mol3, type = 'sdf')
sdffp1 = extractDrugOBFP2(mol4, type = 'sdf')
## End(Not run)
```

extractDrugOBFP3

*Calculate the FP3 Molecular Fingerprints***Description**

Calculate the FP3 Molecular Fingerprints

Usage

```
extractDrugOBFP3(molecules, type = c("smile", "sdf"))
```

Arguments

molecules	R character string object containing the molecules. See the example section for details.
type	'smile' or 'sdf'.

Details

Calculate the 64 bit FP3 fingerprints provided by OpenBabel.

Value

A matrix. Each row represents one molecule, the columns represent the fingerprints.

Examples

```
mol1 = 'C1CCC1CC(CN(C)(C))CC(=O)CC' # one molecule SMILE in a vector
mol2 = c('CCC', 'CCN', 'CCN(C)(C)', 'c1ccccc1Cc1ccccc1',
        'C1CCC1CC(CN(C)(C))CC(=O)CC') # multiple SMILES in a vector
mol3 = readChar(system.file('compseq/DB00860.sdf', package = 'Rcpi'),
                nchars = 1e+6) # single molecule in a sdf file
mol4 = readChar(system.file('sysdata/OptAA3d.sdf', package = 'Rcpi'),
                nchars = 1e+6) # multiple molecules in a sdf file

## Not run:
smifp0 = extractDrugOBFP3(mol1, type = 'smile')
smifp1 = extractDrugOBFP3(mol2, type = 'smile')
```

```
sdffp0 = extractDrugOBFP3(mol3, type = 'sdf')
sdffp1 = extractDrugOBFP3(mol4, type = 'sdf')
## End(Not run)
```

extractDrugOBFP4*Calculate the FP4 Molecular Fingerprints*

Description

Calculate the FP4 Molecular Fingerprints

Usage

```
extractDrugOBFP4(molecules, type = c("smile", "sdf"))
```

Arguments

molecules	R character string object containing the molecules. See the example section for details.
type	'smile' or 'sdf'.

Details

Calculate the 512 bit FP4 fingerprints provided by OpenBabel.

Value

A matrix. Each row represents one molecule, the columns represent the fingerprints.

Examples

```
mol1 = 'C1CCC1CC(CN(C)(C))CC(=O)CC' # one molecule SMILE in a vector
mol2 = c('CCC', 'CCN', 'CCN(C)(C)', 'c1ccccc1Cc1ccccc1',
        'C1CCC1CC(CN(C)(C))CC(=O)CC') # multiple SMILES in a vector
mol3 = readChar(system.file('compseq/DB00860.sdf', package = 'Rcpi'),
                nchars = 1e+6) # single molecule in a sdf file
mol4 = readChar(system.file('sysdata/OptAA3d.sdf', package = 'Rcpi'),
                nchars = 1e+6) # multiple molecules in a sdf file

## Not run:
smifp0 = extractDrugOBFP4(mol1, type = 'smile')
smifp1 = extractDrugOBFP4(mol2, type = 'smile')
sdffp0 = extractDrugOBFP4(mol3, type = 'sdf')
sdffp1 = extractDrugOBFP4(mol4, type = 'sdf')
## End(Not run)
```

extractDrugOBMACCS	<i>Calculate the MACCS Molecular Fingerprints</i>
--------------------	---

Description

Calculate the MACCS Molecular Fingerprints

Usage

```
extractDrugOBMACCS(molecules, type = c("smile", "sdf"))
```

Arguments

molecules	R character string object containing the molecules. See the example section for details.
type	'smile' or 'sdf'.

Details

Calculate the 256 bit MACCS fingerprints provided by OpenBabel.

Value

A matrix. Each row represents one molecule, the columns represent the fingerprints.

Examples

```
mol1 = 'C1CCC1CC(CN(C)(C))CC(=O)CC' # one molecule SMILE in a vector
mol2 = c('CCC', 'CCN', 'CCN(C)(C)', 'c1ccccc1Cc1ccccc1',
        'C1CCC1CC(CN(C)(C))CC(=O)CC') # multiple SMILES in a vector
mol3 = readChar(system.file('compseq/DB00860.sdf', package = 'Rcpi'),
                nchars = 1e+6) # single molecule in a sdf file
mol4 = readChar(system.file('sysdata/OptAA3d.sdf', package = 'Rcpi'),
                nchars = 1e+6) # multiple molecules in a sdf file

## Not run:
# MACCS may not be available in current version of ChemmineOB
smifp0 = extractDrugOBMACCS(mol1, type = 'smile')
smifp1 = extractDrugOBMACCS(mol2, type = 'smile')
sdffp0 = extractDrugOBMACCS(mol3, type = 'sdf')
sdffp1 = extractDrugOBMACCS(mol4, type = 'sdf')
## End(Not run)
```

`extractDrugPetitjeanNumber`*Descriptor that Calculates the Petitjean Number of a Molecule*

Description

Descriptor that Calculates the Petitjean Number of a Molecule

Usage

```
extractDrugPetitjeanNumber(molecules, silent = TRUE)
```

Arguments

<code>molecules</code>	Parsed molecule object.
<code>silent</code>	Logical. Whether the calculating process should be shown or not, default is TRUE.

Details

This descriptor calculates the Petitjean number of a molecule. According to the Petitjean definition, the eccentricity of a vertex corresponds to the distance from that vertex to the most remote vertex in the graph.

The distance is obtained from the distance matrix as the count of edges between the two vertices. If $r(i)$ is the largest matrix entry in row i of the distance matrix D , then the radius is defined as the smallest of the $r(i)$. The graph diameter D is defined as the largest vertex eccentricity in the graph. (<http://www.edusoft-lc.com/molconn/manuals/400/chaptwo.html>)

Value

A data frame, each row represents one of the molecules, each column represents one feature. This function returns one column named `PetitjeanNumber`.

Examples

```
smi = system.file('vignettesdata/FDAMDD.smi', package = 'Rcpi')

mol = readMolFromSmi(smi, type = 'mol')
dat = extractDrugPetitjeanNumber(mol)
head(dat)
```

`extractDrugPetitjeanShapeIndex`*Descriptor that Calculates the Petitjean Shape Indices*

Description

Descriptor that Calculates the Petitjean Shape Indices

Usage

```
extractDrugPetitjeanShapeIndex(molecules, silent = TRUE)
```

Arguments

<code>molecules</code>	Parsed molecule object.
<code>silent</code>	Logical. Whether the calculating process should be shown or not, default is TRUE.

Details

The topological and geometric shape indices described Petitjean and Bath et al. respectively. Both measure the anisotropy in a molecule.

Value

A data frame, each row represents one of the molecules, each column represents one feature. This function returns two columns named `topoShape` (Topological Shape Index) and `geomShape` (Geometric Shape Index).

References

Petitjean, M., Applications of the radius-diameter diagram to the classification of topological and geometrical shapes of chemical compounds, *Journal of Chemical Information and Computer Science*, 1992, 32:331-337

Bath, P.A. and Poirette, A.R. and Willet, P. and Allen, F.H. , The Extent of the Relationship between the Graph-Theoretical and the Geometrical Shape Coefficients of Chemical Compounds, *Journal of Chemical Information and Computer Science*, 1995, 35:714-716.

Examples

```
sdf = system.file('sysdata/OptAA3d.sdf', package = 'Rcpi')

mol = readMolFromSDF(sdf)
dat = extractDrugPetitjeanShapeIndex(mol)
head(dat)
```

extractDrugPubChem	<i>Calculate the PubChem Molecular Fingerprints (in Compact Format)</i>
--------------------	---

Description

Calculate the PubChem Molecular Fingerprints (in Compact Format)

Usage

```
extractDrugPubChem(molecules, silent = TRUE)
```

Arguments

molecules	Parsed molecule object.
silent	Logical. Whether the calculating process should be shown or not, default is TRUE.

Details

Calculate the 881 bit fingerprints defined by PubChem.

Value

A list, each component represents one of the molecules, each element in the component represents the index of which element in the fingerprint is 1. Each component's name is the length of the fingerprints.

See Also

[extractDrugPubChemComplete](#)

Examples

```
smi = system.file('vignettesdata/FDAMDD.smi', package = 'Rcpi')  
  
mol = readMolFromSmi(smi, type = 'mol')  
fp = extractDrugPubChem(mol)  
head(fp)
```

extractDrugPubChemComplete	<i>Calculate the PubChem Molecular Fingerprints (in Complete Format)</i>
----------------------------	--

Description

Calculate the PubChem Molecular Fingerprints (in Complete Format)

Usage

```
extractDrugPubChemComplete(molecules, silent = TRUE)
```

Arguments

molecules	Parsed molecule object.
silent	Logical. Whether the calculating process should be shown or not, default is TRUE.

Details

Calculate the 881 bit fingerprints defined by PubChem.

Value

An integer vector or a matrix. Each row represents one molecule, the columns represent the fingerprints.

See Also

[extractDrugPubChem](#)

Examples

```
smi = system.file('vignettesdata/FDAMDD.smi', package = 'Rcpi')  
  
mol = readMolFromSmi(smi, type = 'mol')  
fp = extractDrugPubChemComplete(mol)  
dim(fp)
```

extractDrugRotatableBondsCount

Descriptor that Calculates the Number of Nonrotatable Bonds on A Molecule

Description

Descriptor that Calculates the Number of Nonrotatable Bonds on A Molecule

Usage

```
extractDrugRotatableBondsCount(molecules, silent = TRUE)
```

Arguments

molecules	Parsed molecule object.
silent	Logical. Whether the calculating process should be shown or not, default is TRUE.

Details

The number of rotatable bonds is given by the SMARTS specified by Daylight on SMARTS tutorial (https://www.daylight.com/dayhtml_tutorials/languages/smarts/smarts_examples.html)

Value

A data frame, each row represents one of the molecules, each column represents one feature. This function returns one column named nRotB.

Examples

```
smi = system.file('vignettesdata/FDAMDD.smi', package = 'Rcpi')

mol = readMolFromSmi(smi, type = 'mol')
dat = extractDrugRotatableBondsCount(mol)
head(dat)
```

extractDrugRuleOfFive *Descriptor that Calculates the Number Failures of the Lipinski's Rule Of Five*

Description

Descriptor that Calculates the Number Failures of the Lipinski's Rule Of Five

Usage

```
extractDrugRuleOfFive(molecules, silent = TRUE)
```

Arguments

molecules	Parsed molecule object.
silent	Logical. Whether the calculating process should be shown or not, default is TRUE.

Details

This descriptor calculates the number failures of the Lipinski's Rule Of Five: http://en.wikipedia.org/wiki/Lipinski%27s_Rule_of_Five.

Value

A data frame, each row represents one of the molecules, each column represents one feature. This function returns one column named LipinskiFailures.

Examples

```
smi = system.file('vignettesdata/FDAMDD.smi', package = 'Rcpi')

mol = readMolFromSmi(smi, type = 'mol')
dat = extractDrugRuleOfFive(mol)
head(dat)
```

`extractDrugShortestPath`*Calculate the Shortest Path Molecular Fingerprints (in Compact Format)*

Description

Calculate the Shortest Path Molecular Fingerprints (in Compact Format)

Usage

```
extractDrugShortestPath(molecules, depth = 6, size = 1024, silent = TRUE)
```

Arguments

<code>molecules</code>	Parsed molecule object.
<code>depth</code>	The search depth. Default is 6.
<code>size</code>	The length of the fingerprint bit string. Default is 1024.
<code>silent</code>	Logical. Whether the calculating process should be shown or not, default is TRUE.

Details

Calculate the fingerprint based on the shortest paths between pairs of atoms and takes into account ring systems, charges etc.

Value

A list, each component represents one of the molecules, each element in the component represents the index of which element in the fingerprint is 1. Each component's name is the length of the fingerprints.

See Also

[extractDrugShortestPathComplete](#)

Examples

```
smi = system.file('vignettesdata/FDAMDD.smi', package = 'Rcpi')  
  
mol = readMolFromSmi(smi, type = 'mol')  
fp = extractDrugShortestPath(mol)  
head(fp)
```

`extractDrugShortestPathComplete`*Calculate the Shortest Path Molecular Fingerprints (in Complete Format)*

Description

Calculate the Shortest Path Molecular Fingerprints (in Complete Format)

Usage

```
extractDrugShortestPathComplete(  
  molecules,  
  depth = 6,  
  size = 1024,  
  silent = TRUE  
)
```

Arguments

<code>molecules</code>	Parsed molecule object.
<code>depth</code>	The search depth. Default is 6.
<code>size</code>	The length of the fingerprint bit string. Default is 1024.
<code>silent</code>	Logical. Whether the calculating process should be shown or not, default is TRUE.

Details

Calculate the fingerprint based on the shortest paths between pairs of atoms and takes into account ring systems, charges etc.

Value

An integer vector or a matrix. Each row represents one molecule, the columns represent the fingerprints.

See Also

[extractDrugShortestPath](#)

Examples

```
smi = system.file('vignettesdata/FDAMDD.smi', package = 'Rcpi')  
  
mol = readMolFromSmi(smi, type = 'mol')  
fp = extractDrugShortestPathComplete(mol)  
dim(fp)
```

extractDrugStandard	<i>Calculate the Standard Molecular Fingerprints (in Compact Format)</i>
---------------------	--

Description

Calculate the Standard Molecular Fingerprints (in Compact Format)

Usage

```
extractDrugStandard(molecules, depth = 6, size = 1024, silent = TRUE)
```

Arguments

molecules	Parsed molecule object.
depth	The search depth. Default is 6.
size	The length of the fingerprint bit string. Default is 1024.
silent	Logical. Whether the calculating process should be shown or not, default is TRUE.

Details

Calculate the standard molecular fingerprints. Considers paths of a given length. This is hashed fingerprints, with a default length of 1024.

Value

A list, each component represents one of the molecules, each element in the component represents the index of which element in the fingerprint is 1. Each component's name is the length of the fingerprints.

See Also

[extractDrugStandardComplete](#)

Examples

```
smi = system.file('vignettesdata/FDAMDD.smi', package = 'Rcpi')  
  
mol = readMolFromSmi(smi, type = 'mol')  
fp = extractDrugStandard(mol)  
head(fp)
```

`extractDrugStandardComplete`*Calculate the Standard Molecular Fingerprints (in Complete Format)*

Description

Calculate the Standard Molecular Fingerprints (in Complete Format)

Usage

```
extractDrugStandardComplete(molecules, depth = 6, size = 1024, silent = TRUE)
```

Arguments

<code>molecules</code>	Parsed molecule object.
<code>depth</code>	The search depth. Default is 6.
<code>size</code>	The length of the fingerprint bit string. Default is 1024.
<code>silent</code>	Logical. Whether the calculating process should be shown or not, default is TRUE.

Details

Calculate the standard molecular fingerprints. Considers paths of a given length. This is hashed fingerprints, with a default length of 1024.

Value

An integer vector or a matrix. Each row represents one molecule, the columns represent the fingerprints.

See Also

[extractDrugStandard](#)

Examples

```
smi = system.file('vignettesdata/FDAMDD.smi', package = 'Rcpi')  
  
mol = readMolFromSmi(smi, type = 'mol')  
fp = extractDrugStandardComplete(mol)  
dim(fp)
```

extractDrugTPSA	<i>Descriptor of Topological Polar Surface Area Based on Fragment Contributions (TPSA)</i>
-----------------	--

Description

Descriptor of Topological Polar Surface Area Based on Fragment Contributions (TPSA)

Usage

```
extractDrugTPSA(molecules, silent = TRUE)
```

Arguments

molecules	Parsed molecule object.
silent	Logical. Whether the calculating process should be shown or not, default is TRUE.

Details

Calculate the descriptor of topological polar surface area based on fragment contributions (TPSA).

Value

A data frame, each row represents one of the molecules, each column represents one feature. This function returns one column named TopoPSA.

References

Ertl, P., Rohde, B., & Selzer, P. (2000). Fast calculation of molecular polar surface area as a sum of fragment-based contributions and its application to the prediction of drug transport properties. *Journal of medicinal chemistry*, 43(20), 3714-3717.

Examples

```
smi = system.file('vignettesdata/FDAMDD.smi', package = 'Rcpi')

mol = readMolFromSmi(smi, type = 'mol')
dat = extractDrugTPSA(mol)
head(dat)
```

extractDrugVABC	<i>Descriptor that Calculates the Volume of A Molecule</i>
-----------------	--

Description

Descriptor that Calculates the Volume of A Molecule

Usage

```
extractDrugVABC(molecules, silent = TRUE)
```

Arguments

molecules	Parsed molecule object.
silent	Logical. Whether the calculating process should be shown or not, default is TRUE.

Details

This descriptor calculates the volume of a molecule.

Value

A data frame, each row represents one of the molecules, each column represents one feature. This function returns one column named VABC.

Examples

```
smi = system.file('vignettesdata/FDAMDD.smi', package = 'Rcpi')  
  
mol = readMolFromSmi(smi, type = 'mol')  
dat = extractDrugVABC(mol)  
head(dat)
```

extractDrugVAdjMa	<i>Descriptor that Calculates the Vertex Adjacency Information of A Molecule</i>
-------------------	--

Description

Descriptor that Calculates the Vertex Adjacency Information of A Molecule

Usage

```
extractDrugVAdjMa(molecules, silent = TRUE)
```

Arguments

molecules	Parsed molecule object.
silent	Logical. Whether the calculating process should be shown or not, default is TRUE.

Details

Vertex adjacency information (magnitude): $1 + \log_2^m$ where m is the number of heavy-heavy bonds. If m is zero, then 0 is returned.

Value

A data frame, each row represents one of the molecules, each column represents one feature. This function returns one column named VAdjMat.

Examples

```
smi = system.file('vignettesdata/FDAMDD.smi', package = 'Rcpi')

mol = readMolFromSmi(smi, type = 'mol')
dat = extractDrugVAdjMa(mol)
head(dat)
```

extractDrugWeight	<i>Descriptor that Calculates the Total Weight of Atoms</i>
-------------------	---

Description

Descriptor that Calculates the Total Weight of Atoms

Usage

```
extractDrugWeight(molecules, silent = TRUE)
```

Arguments

molecules	Parsed molecule object.
silent	Logical. Whether the calculating process should be shown or not, default is TRUE.

Details

This descriptor calculates the molecular weight.

Value

A data frame, each row represents one of the molecules, each column represents one feature. This function returns one column named MW.

Examples

```
smi = system.file('vignettesdata/FDAMDD.smi', package = 'Rcpi')

mol = readMolFromSmi(smi, type = 'mol')
dat = extractDrugWeight(mol)
head(dat)
```

`extractDrugWeightedPath`*Descriptor that Calculates the Weighted Path (Molecular ID)*

Description

Descriptor that Calculates the Weighted Path (Molecular ID)

Usage

```
extractDrugWeightedPath(molecules, silent = TRUE)
```

Arguments

<code>molecules</code>	Parsed molecule object.
<code>silent</code>	Logical. Whether the calculating process should be shown or not, default is TRUE.

Details

This descriptor calculates the weighted path (molecular ID) described by Randic, characterizing molecular branching. Five descriptors are calculated, based on the implementation in the ADAPT software package. Note that the descriptor is based on identifying all paths between pairs of atoms and so is NP-hard. This means that it can take some time for large, complex molecules.

Value

A data frame, each row represents one of the molecules, each column represents one feature. This function returns 5 columns named WTPT.1, WTPT.2, WTPT.3, WTPT.4, WTPT.5:

- WTPT.1 - molecular ID
- WTPT.2 - molecular ID / number of atoms
- WTPT.3 - sum of path lengths starting from heteroatoms
- WTPT.4 - sum of path lengths starting from oxygens
- WTPT.5 - sum of path lengths starting from nitrogens

References

Randic, M., On molecular identification numbers (1984). Journal of Chemical Information and Computer Science, 24:164-175.

Examples

```
smi = system.file('vignettesdata/FDAMDD.smi', package = 'Rcpi')

mol = readMolFromSmi(smi, type = 'mol')
dat = extractDrugWeightedPath(mol)
head(dat)
```

extractDrugWHIM

Calculate Holistic Descriptors Described by Todeschini et al.

Description

Calculate Holistic Descriptors Described by Todeschini et al.

Usage

```
extractDrugWHIM(molecules, silent = TRUE)
```

Arguments

molecules	Parsed molecule object.
silent	Logical. Whether the calculating process should be shown or not, default is TRUE.

Details

Holistic descriptors described by Todeschini et al, the descriptors are based on a number of atom weightings. There are six different possible weightings:

- unit weights
- atomic masses
- van der Waals volumes
- Mulliken atomic electronegativities
- atomic polarizabilities
- E-state values described by Kier and Hall

Currently weighting schemes 1, 2, 3, 4 and 5 are implemented. The weight values are taken from Todeschini et al. and as a result 19 elements are considered. For each weighting scheme we can obtain

- 11 directional WHIM descriptors (lambda1 .. 3, nu1 .. 2, gamma1 .. 3, eta1 .. 3)
- 6 non-directional WHIM descriptors (T, A, V, K, G, D)

Though Todeschini et al. mentions that for planar molecules only 8 directional WHIM descriptors are required the current code will return all 11.

Value

A data frame, each row represents one of the molecules, each column represents one feature. This function returns 17 columns:

- Wlambda1
- Wlambda2
- wlambda3
- Wnu1
- Wnu2

- Wgamma1
- Wgamma2
- Wgamma3
- Weta1
- Weta2
- Weta3
- WT
- WA
- WV
- WK
- WG
- WD

Each name will have a suffix of the form .X where X indicates the weighting scheme used. Possible values of X are

- unity
- mass
- volume
- eneg
- polar

References

Todeschini, R. and Gramatica, P., New 3D Molecular Descriptors: The WHIM theory and QAR Applications, *Persepectives in Drug Discovery and Design*, 1998, 7:355-380.

Examples

```
sdf = system.file('sysdata/OptAA3d.sdf', package = 'Rcpi')  
  
mol = readMolFromSDF(sdf)  
dat = extractDrugWHIM(mol)  
head(dat)
```

extractDrugWienerNumbers

Descriptor that Calculates Wiener Path Number and Wiener Polarity Number

Description

Descriptor that Calculates Wiener Path Number and Wiener Polarity Number

Usage

```
extractDrugWienerNumbers(molecules, silent = TRUE)
```

Arguments

molecules	Parsed molecule object.
silent	Logical. Whether the calculating process should be shown or not, default is TRUE.

Details

This descriptor calculates the Wiener numbers, including the Wiener Path number and the Wiener Polarity Number. Wiener path number: half the sum of all the distance matrix entries; Wiener polarity number: half the sum of all the distance matrix entries with a value of 3.

Value

A data frame, each row represents one of the molecules, each column represents one feature. This function returns two columns named WPATH (weiner path number) and WPOL (weiner polarity number).

References

Wiener, H. (1947). Structural determination of paraffin boiling points. Journal of the American Chemical Society, 69(1), 17-20.

Examples

```
smi = system.file('vignettesdata/FDAMDD.smi', package = 'Rcpi')  
  
mol = readMolFromSmi(smi, type = 'mol')  
dat = extractDrugWienerNumbers(mol)  
head(dat)
```

extractDrugXLogP	<i>Descriptor that Calculates the Prediction of logP Based on the Atom-Type Method Called XLogP</i>
------------------	---

Description

Descriptor that Calculates the Prediction of logP Based on the Atom-Type Method Called XLogP

Usage

```
extractDrugXLogP(molecules, silent = TRUE)
```

Arguments

molecules	Parsed molecule object.
silent	Logical. Whether the calculating process should be shown or not, default is TRUE.

Details

Prediction of logP based on the atom-type method called XLogP.

Value

A data frame, each row represents one of the molecules, each column represents one feature. This function returns one column named XLogP.

References

Wang, R., Fu, Y., and Lai, L., A New Atom-Additive Method for Calculating Partition Coefficients, Journal of Chemical Information and Computer Sciences, 1997, 37:615-621.

Wang, R., Gao, Y., and Lai, L., Calculating partition coefficient by atom-additive method, Perspectives in Drug Discovery and Design, 2000, 19:47-66.

Examples

```
smi = system.file('vignettesdata/FDAMDD.smi', package = 'Rcpi')

mol = readMolFromSmi(smi, type = 'mol')
dat = extractDrugXLogP(mol)
head(dat)
```

extractDrugZagrebIndex

Descriptor that Calculates the Sum of the Squared Atom Degrees of All Heavy Atoms

Description

Descriptor that Calculates the Sum of the Squared Atom Degrees of All Heavy Atoms

Usage

```
extractDrugZagrebIndex(molecules, silent = TRUE)
```

Arguments

molecules	Parsed molecule object.
silent	Logical. Whether the calculating process should be shown or not, default is TRUE.

Details

Zagreb index: the sum of the squares of atom degree over all heavy atoms i .

Value

A data frame, each row represents one of the molecules, each column represents one feature. This function returns one column named Zagreb.

Examples

```
smi = system.file('vignettedata/FDAMDD.smi', package = 'Rcpi')

mol = readMolFromSmi(smi, type = 'mol')
dat = extractDrugZagrebIndex(mol)
head(dat)
```

extractPCMBLOSUM

*Generalized BLOSUM and PAM Matrix-Derived Descriptors***Description**

Generalized BLOSUM and PAM Matrix-Derived Descriptors

Usage

```
extractPCMBLOSUM(x, submat = "AABLOSUM62", k, lag, scale = TRUE, silent = TRUE)
```

Arguments

x	A character vector, as the input protein sequence.
submat	Substitution matrix for the 20 amino acids. Should be one of AABLOSUM45, AABLOSUM50, AABLOSUM62, AABLOSUM80, AABLOSUM100, AAPAM30, AAPAM40, AAPAM70, AAPAM120, AAPAM250. Default is 'AABLOSUM62'.
k	Integer. The number of selected scales (i.e. the first k scales) derived by the substitution matrix. This could be selected according to the printed relative importance values.
lag	The lag parameter. Must be less than the amino acids.
scale	Logical. Should we auto-scale the substitution matrix (submat) before doing eigen decomposition? Default is TRUE.
silent	Logical. Whether we print the relative importance of each scales (diagonal value of the eigen decomposition result matrix B) or not. Default is TRUE.

Details

This function calculates the generalized BLOSUM matrix-derived descriptors. For users' convenience, Rcpi provides the BLOSUM45, BLOSUM50, BLOSUM62, BLOSUM80, BLOSUM100, PAM30, PAM40, PAM70, PAM120, and PAM250 matrices for the 20 amino acids to select.

Value

A length $\text{lag} * p^2$ named vector, p is the number of scales selected.

References

Georgiev, A. G. (2009). Interpretable numerical descriptors of amino acid space. *Journal of Computational Biology*, 16(5), 703–723.

Examples

```
x = readFASTA(system.file('protseq/P00750.fasta', package = 'Rcpi'))[[1]]
blosum = extractPCMBLOSUM(x, submat = 'AABLOSUM62', k = 5, lag = 7, scale = TRUE, silent = FALSE)
```

extractPCMDescScales *Scales-Based Descriptors with 20+ classes of Molecular Descriptors*

Description

Scales-Based Descriptors with 20+ classes of Molecular Descriptors

Usage

```
extractPCMDescScales(
  x,
  propmat,
  index = NULL,
  pc,
  lag,
  scale = TRUE,
  silent = TRUE
)
```

Arguments

x	A character vector, as the input protein sequence.
propmat	The matrix containing the descriptor set for the amino acids, which could be chosen from AAMOE2D, AAMOE3D, AACPSA, AADescAll, AA2DACOR, AA3DMoRSE, AAACF, AABurden, AAConn, AAConst, AAEdgeAdj, AA EigIdx, AAFGC, AAGeom, AAGETAWAY, AAInfo, AAMolProp, AARandic, AARDF, AATopo, AATopoChg, AAWalk, AAWHIM.
index	Integer vector or character vector. Specify which molecular descriptors to select from one of these descriptor sets by specify the numerical or character index of the molecular descriptors in the descriptor set. Default is NULL, means selecting all the molecular descriptors in this descriptor set.
pc	Integer. The maximum dimension of the space which the data are to be represented in. Must be no greater than the number of AA properties provided.
lag	The lag parameter. Must be less than the amino acids.
scale	Logical. Should we auto-scale the property matrix (propmat) before doing MDS? Default is TRUE.
silent	Logical. Whether we print the standard deviation, proportion of variance and the cumulative proportion of the selected principal components or not. Default is TRUE.

Details

This function calculates the scales-based descriptors with molecular descriptors sets calculated by Dragon, Discovery Studio and MOE. Users could specify which molecular descriptors to select from one of these descriptor sets by specify the numerical or character index of the molecular descriptors in the descriptor set.

Value

A length lag * p^2 named vector, p is the number of scales selected.

See Also

See [extractPCMScales](#) for generalized AA-descriptor based scales descriptors.

Examples

```
x = readFASTA(system.file('protseq/P00750.fasta', package = 'Rcpi'))[[1]]
descscales = extractPCMDescScales(x, propmat = 'AATopo', index = c(37:41, 43:47),
                                  pc = 5, lag = 7, silent = FALSE)
```

extractPCMFAScales	<i>Generalized Scales-Based Descriptors derived by Factor Analysis</i>
--------------------	--

Description

Generalized Scales-Based Descriptors derived by Factor Analysis

Usage

```
extractPCMFAScales(
  x,
  propmat,
  factors,
  scores = "regression",
  lag,
  scale = TRUE,
  silent = TRUE
)
```

Arguments

x	A character vector, as the input protein sequence.
propmat	A matrix containing the properties for the amino acids. Each row represent one amino acid type, each column represents one property. Note that the one-letter row names must be provided for we need them to seek the properties for each AA type.
factors	Integer. The number of factors to be fitted. Must be no greater than the number of AA properties provided.
scores	Type of scores to produce. The default is "regression", which gives Thompson's scores, "Bartlett" given Bartlett's weighted least-squares scores.
lag	The lag parameter. Must be less than the amino acids number in the protein sequence.
scale	Logical. Should we auto-scale the property matrix (propmat) before doing Factor Analysis? Default is TRUE.
silent	Logical. Whether we print the SS loadings, proportion of variance and the cumulative proportion of the selected factors or not. Default is TRUE.

Details

This function calculates the generalized scales-based descriptors derived by Factor Analysis (FA). Users could provide customized amino acid property matrices.

Value

A length $\text{lag} * p^2$ named vector, p is the number of scales (factors) selected.

References

Atchley, W. R., Zhao, J., Fernandes, A. D., & Druke, T. (2005). Solving the protein sequence metric problem. *Proceedings of the National Academy of Sciences of the United States of America*, 102(18), 6395-6400.

Examples

```
x = readFASTA(system.file('protseq/P00750.fasta', package = 'Rcpi'))[[1]]
data(AATopo)
tprops = AATopo[, c(37:41, 43:47)] # select a set of topological descriptors
fa = extractPCMFAScales(x, propmat = tprops, factors = 5, lag = 7, silent = FALSE)
```

extractPCMDSScales	<i>Generalized Scales-Based Descriptors derived by Multidimensional Scaling</i>
--------------------	---

Description

Generalized Scales-Based Descriptors derived by Multidimensional Scaling

Usage

```
extractPCMDSScales(x, propmat, k, lag, scale = TRUE, silent = TRUE)
```

Arguments

<code>x</code>	A character vector, as the input protein sequence.
<code>propmat</code>	A matrix containing the properties for the amino acids. Each row represent one amino acid type, each column represents one property. Note that the one-letter row names must be provided for we need them to seek the properties for each AA type.
<code>k</code>	Integer. The maximum dimension of the space which the data are to be represented in. Must be no greater than the number of AA properties provided.
<code>lag</code>	The lag parameter. Must be less than the amino acids.
<code>scale</code>	Logical. Should we auto-scale the property matrix (propmat) before doing MDS? Default is TRUE.
<code>silent</code>	Logical. Whether we print the k eigenvalues computed during the scaling process or not. Default is TRUE.

Details

This function calculates the generalized scales-based descriptors derived by Multidimensional Scaling (MDS). Users could provide customized amino acid property matrices.

Value

A length $\text{lag} * p^2$ named vector, p is the number of scales (dimensionality) selected.

References

Venkatarajan, M. S., & Braun, W. (2001). New quantitative descriptors of amino acids based on multidimensional scaling of a large number of physical-chemical properties. *Molecular modeling annual*, 7(12), 445–453.

See Also

See [extractPCMScales](#) for generalized scales-based descriptors derived by Principal Components Analysis.

Examples

```
x = readFASTA(system.file('protseq/P00750.fasta', package = 'Rcpi'))[[1]]
data(AATopo)
tprops = AATopo[, c(37:41, 43:47)] # select a set of topological descriptors
mds = extractPCMDSScales(x, propmat = tprops, k = 5, lag = 7, silent = FALSE)
```

extractPCMPPropScales *Generalized AA-Properties Based Scales Descriptors*

Description

Generalized AA-Properties Based Scales Descriptors

Usage

```
extractPCMPPropScales(x, index = NULL, pc, lag, scale = TRUE, silent = TRUE)
```

Arguments

<code>x</code>	A character vector, as the input protein sequence.
<code>index</code>	Integer vector or character vector. Specify which AAindex properties to select from the AAindex database by specify the numerical or character index of the properties in the AAindex database. Default is NULL, means selecting all the AA properties in the AAindex database.
<code>pc</code>	Integer. Use the first pc principal components as the scales. Must be no greater than the number of AA properties provided.
<code>lag</code>	The lag parameter. Must be less than the amino acids.
<code>scale</code>	Logical. Should we auto-scale the property matrix before PCA? Default is TRUE.
<code>silent</code>	Logical. Whether we print the standard deviation, proportion of variance and the cumulative proportion of the selected principal components or not. Default is TRUE.

Details

This function calculates the generalized amino acid properties based scales descriptors. Users could specify which AAindex properties to select from the AAindex database by specify the numerical or character index of the properties in the AAindex database.

Value

A length $\text{lag} * p^2$ named vector, p is the number of scales (principal components) selected.

See Also

See [extractPCMScales](#) for generalized scales-based descriptors.

Examples

```
x = readFASTA(system.file('protseq/P00750.fasta', package = 'Rcpi'))[[1]]
propscales = extractPCMPPropScales(x, index = c(160:165, 258:296), pc = 5, lag = 7, silent = FALSE)
```

extractPCMScales	<i>Generalized Scales-Based Descriptors derived by Principal Components Analysis</i>
------------------	--

Description

Generalized Scales-Based Descriptors derived by Principal Components Analysis

Usage

```
extractPCMScales(x, propmat, pc, lag, scale = TRUE, silent = TRUE)
```

Arguments

x	A character vector, as the input protein sequence.
propmat	A matrix containing the properties for the amino acids. Each row represent one amino acid type, each column represents one property. Note that the one-letter row names must be provided for we need them to seek the properties for each AA type.
pc	Integer. Use the first pc principal components as the scales. Must be no greater than the number of AA properties provided.
lag	The lag parameter. Must be less than the amino acids.
scale	Logical. Should we auto-scale the property matrix (propmat) before PCA? Default is TRUE.
silent	Logical. Whether we print the standard deviation, proportion of variance and the cumulative proportion of the selected principal components or not. Default is TRUE.

Details

This function calculates the generalized scales-based descriptors derived by Principal Components Analysis (PCA). Users could provide customized amino acid property matrices. This function implements the core computation procedure needed for the generalized scales-based descriptors derived by AA-Properties (AAindex) and generalized scales-based descriptors derived by 20+ classes of 2D and 3D molecular descriptors (Topological, WHIM, VHSE, etc.).

Value

A length $\text{lag} * p^2$ named vector, p is the number of scales (principal components) selected.

See Also

See [extractPCMDescScales](#) for generalized AA property based scales descriptors, and [extractPCMPPropScales](#) for (19 classes) AA descriptor based scales descriptors.

Examples

```
x = readFASTA(system.file('protseq/P00750.fasta', package = 'Rcpi'))[[1]]
data(AAindex)
AAidxmat = t(na.omit(as.matrix(AAindex[, 7:26])))
scales = extractPCMScales(x, propmat = AAidxmat, pc = 5, lag = 7, silent = FALSE)
```

extractProtAAC	<i>Amino Acid Composition Descriptor</i>
----------------	--

Description

Amino Acid Composition Descriptor

Usage

```
extractProtAAC(x)
```

Arguments

x A character vector, as the input protein sequence.

Details

This function calculates the Amino Acid Composition descriptor (Dim: 20).

Value

A length 20 named vector

References

M. Bhasin, G. P. S. Raghava. Classification of Nuclear Receptors Based on Amino Acid Composition and Dipeptide Composition. *Journal of Biological Chemistry*, 2004, 279, 23262.

See Also

See [extractProtDC](#) and [extractProtTC](#) for Dipeptide Composition and Tripeptide Composition descriptors.

Examples

```
x = readFASTA(system.file('protseq/P00750.fasta', package = 'Rcpi'))[[1]]
extractProtAAC(x)
```

extractProtAPAAC	<i>Amphiphilic Pseudo Amino Acid Composition Descriptor</i>
------------------	---

Description

Amphiphilic Pseudo Amino Acid Composition Descriptor

Usage

```
extractProtAPAAC(
  x,
  props = c("Hydrophobicity", "Hydrophilicity"),
  lambda = 30,
  w = 0.05,
  customprops = NULL
)
```

Arguments

x	A character vector, as the input protein sequence.
props	A character vector, specifying the properties used. 2 properties are used by default, as listed below: 'Hydrophobicity' Hydrophobicity value of the 20 amino acids 'Hydrophilicity' Hydrophilicity value of the 20 amino acids
lambda	The lambda parameter for the APAAC descriptors, default is 30.
w	The weighting factor, default is 0.05.
customprops	A n x 21 named data frame contains n customize property. Each row contains one property. The column order for different amino acid types is 'AccNo', 'A', 'R', 'N', 'D', 'C', 'E', 'Q', 'G', 'H', 'I', 'L', 'K', 'M', 'F', 'P', 'S', 'T', 'W', 'Y', 'V', and the columns should also be <i>exactly</i> named like this. The AccNo column contains the properties' names. Then users should explicitly specify these properties with these names in the argument props. See the examples below for a demonstration. The default value for customprops is NULL.

Details

This function calculates the Amphiphilic Pseudo Amino Acid Composition (APAAC) descriptor (Dim: 20 + (n * lambda), n is the number of properties selected, default is 80).

extractProtCTDC	<i>CTD Descriptors - Composition</i>
-----------------	--------------------------------------

Description

CTD Descriptors - Composition

Usage

```
extractProtCTDC(x)
```

Arguments

x A character vector, as the input protein sequence.

Details

This function calculates the Composition descriptor of the CTD descriptors (Dim: 21).

Value

A length 21 named vector

References

Inna Dubchak, Ilya Muchink, Stephen R. Holbrook and Sung-Hou Kim. Prediction of protein folding class using global description of amino acid sequence. *Proceedings of the National Academy of Sciences*. USA, 1995, 92, 8700-8704.

Inna Dubchak, Ilya Muchink, Christopher Mayor, Igor Dralyuk and Sung-Hou Kim. Recognition of a Protein Fold in the Context of the SCOP classification. *Proteins: Structure, Function and Genetics*, 1999, 35, 401-407.

See Also

See [extractProtCTDT](#) and [extractProtCTDD](#) for the Transition and Distribution descriptors.

Examples

```
x = readFASTA(system.file('protseq/P00750.fasta', package = 'Rcpi'))[[1]]
extractProtCTDC(x)
```

extractProtCTDD	<i>CTD Descriptors - Distribution</i>
-----------------	---------------------------------------

Description

CTD Descriptors - Distribution

Usage

```
extractProtCTDD(x)
```

Arguments

x A character vector, as the input protein sequence.

Details

This function calculates the Distribution descriptor of the CTD descriptors (Dim: 105).

Value

A length 105 named vector

References

Inna Dubchak, Ilya Muchink, Stephen R. Holbrook and Sung-Hou Kim. Prediction of protein folding class using global description of amino acid sequence. *Proceedings of the National Academy of Sciences*. USA, 1995, 92, 8700-8704.

Inna Dubchak, Ilya Muchink, Christopher Mayor, Igor Dralyuk and Sung-Hou Kim. Recognition of a Protein Fold in the Context of the SCOP classification. *Proteins: Structure, Function and Genetics*, 1999, 35, 401-407.

See Also

See [extractProtCTDC](#) and [extractProtCTDT](#) for the Composition and Transition descriptors.

Examples

```
x = readFASTA(system.file('protseq/P00750.fasta', package = 'Rcpi'))[[1]]
extractProtCTDD(x)
```

extractProtCTDT	<i>CTD Descriptors - Transition</i>
-----------------	-------------------------------------

Description

CTD Descriptors - Transition

Usage

```
extractProtCTDT(x)
```

Arguments

x A character vector, as the input protein sequence.

Details

This function calculates the Transition descriptor of the CTD descriptors (Dim: 21).

Value

A length 21 named vector

References

Inna Dubchak, Ilya Muchink, Stephen R. Holbrook and Sung-Hou Kim. Prediction of protein folding class using global description of amino acid sequence. *Proceedings of the National Academy of Sciences*. USA, 1995, 92, 8700-8704.

Inna Dubchak, Ilya Muchink, Christopher Mayor, Igor Dralyuk and Sung-Hou Kim. Recognition of a Protein Fold in the Context of the SCOP classification. *Proteins: Structure, Function and Genetics*, 1999, 35, 401-407.

See Also

See [extractProtCTDC](#) and [extractProtCTDD](#) for the Composition and Distribution descriptors.

Examples

```
x = readFASTA(system.file('protseq/P00750.fasta', package = 'Rcpi'))[[1]]
extractProtCTDT(x)
```

extractProtCTriad	<i>Conjoint Triad Descriptor</i>
-------------------	----------------------------------

Description

Conjoint Triad Descriptor

Usage

```
extractProtCTriad(x)
```

Arguments

x A character vector, as the input protein sequence.

Details

This function calculates the Conjoint Triad descriptor (Dim: 343).

Value

A length 343 named vector

References

J.W. Shen, J. Zhang, X.M. Luo, W.L. Zhu, K.Q. Yu, K.X. Chen, Y.X. Li, H.L. Jiang. Predicting Protein-protein Interactions Based Only on Sequences Information. *Proceedings of the National Academy of Sciences*. 007, 104, 4337–4341.

Examples

```
x = readFASTA(system.file('protseq/P00750.fasta', package = 'Rcpi'))[[1]]
extractProtCTriad(x)
```

extractProtDC	<i>Dipeptide Composition Descriptor</i>
---------------	---

Description

Dipeptide Composition Descriptor

Usage

```
extractProtDC(x)
```

Arguments

x A character vector, as the input protein sequence.

Details

This function calculates the Dipeptide Composition descriptor (Dim: 400).

Value

A length 400 named vector

References

M. Bhasin, G. P. S. Raghava. Classification of Nuclear Receptors Based on Amino Acid Composition and Dipeptide Composition. *Journal of Biological Chemistry*, 2004, 279, 23262.

See Also

See [extractProtAAC](#) and [extractProtTC](#) for Amino Acid Composition and Tripeptide Composition descriptors.

Examples

```
x = readFASTA(system.file('protseq/P00750.fasta', package = 'Rcpi'))[[1]]
extractProtDC(x)
```

extractProtGeary

Geary Autocorrelation Descriptor

Description

Geary Autocorrelation Descriptor

Usage

```
extractProtGeary(
  x,
  props = c("CIDH920105", "BHAR880101", "CHAM820101", "CHAM820102", "CHOC760101",
    "BIGC670101", "CHAM810101", "DAYM780201"),
  nlag = 30L,
  customprops = NULL
)
```

Arguments

x	A character vector, as the input protein sequence.
props	A character vector, specifying the Accession Number of the target properties. 8 properties are used by default, as listed below: AccNo. CIDH920105 Normalized average hydrophobicity scales (Cid et al., 1992) AccNo. BHAR880101 Average flexibility indices (Bhaskaran-Ponnuswamy, 1988) AccNo. CHAM820101 Polarizability parameter (Charton-Charton, 1982) AccNo. CHAM820102 Free energy of solution in water, kcal/mole (Charton-Charton, 1982)

	AccNo. CHOC760101 Residue accessible surface area in tripeptide (Chothia, 1976)
	AccNo. BIGC670101 Residue volume (Bigelow, 1967)
	AccNo. CHAM810101 Steric parameter (Charton, 1981)
	AccNo. DAYM780201 Relative mutability (Dayhoff et al., 1978b)
nlag	Maximum value of the lag parameter. Default is 30.
customprops	A n x 21 named data frame contains n customize property. Each row contains one property. The column order for different amino acid types is 'AccNo', 'A', 'R', 'N', 'D', 'C', 'E', 'Q', 'G', 'H', 'I', 'L', 'K', 'M', 'F', 'P', 'S', 'T', 'W', 'Y', 'V', and the columns should also be <i>exactly</i> named like this. The AccNo column contains the properties' names. Then users should explicitly specify these properties with these names in the argument props. See the examples below for a demonstration. The default value for customprops is NULL.

Details

This function calculates the Geary autocorrelation descriptor (Dim: length(props) * nlag).

Value

A length nlag named vector

References

- AAindex: Amino acid index database. <https://www.genome.jp/dbget/aaindex.html>
- Feng, Z.P. and Zhang, C.T. (2000) Prediction of membrane protein types based on the hydrophobic index of amino acids. *Journal of Protein Chemistry*, 19, 269-275.
- Horne, D.S. (1988) Prediction of protein helix content from an autocorrelation analysis of sequence hydrophobicities. *Biopolymers*, 27, 451-477.
- Sokal, R.R. and Thomson, B.A. (2006) Population structure inferred by local spatial autocorrelation: an usage from an Amerindian tribal population. *American Journal of Physical Anthropology*, 129, 121-131.

See Also

See [extractProtMoreauBroto](#) and [extractProtMoran](#) for Moreau-Broto autocorrelation descriptors and Moran autocorrelation descriptors.

Examples

```
x = readFASTA(system.file('protseq/P00750.fasta', package = 'Rcpi'))[[1]]
extractProtGeary(x)

myprops = data.frame(AccNo = c("MyProp1", "MyProp2", "MyProp3"),
  A = c(0.62, -0.5, 15), R = c(-2.53, 3, 101),
  N = c(-0.78, 0.2, 58), D = c(-0.9, 3, 59),
  C = c(0.29, -1, 47), E = c(-0.74, 3, 73),
  Q = c(-0.85, 0.2, 72), G = c(0.48, 0, 1),
  H = c(-0.4, -0.5, 82), I = c(1.38, -1.8, 57),
  L = c(1.06, -1.8, 57), K = c(-1.5, 3, 73),
  M = c(0.64, -1.3, 75), F = c(1.19, -2.5, 91),
  P = c(0.12, 0, 42), S = c(-0.18, 0.3, 31),
  T = c(-0.05, -0.4, 45), W = c(0.81, -3.4, 130),
```

```

Y = c(0.26, -2.3, 107), V = c(1.08, -1.5, 43))

# Use 4 properties in the AAindex database, and 3 customized properties
extractProtGeary(x, customprops = myprops,
  props = c('CIDH920105', 'BHAR880101',
            'CHAM820101', 'CHAM820102',
            'MyProp1', 'MyProp2', 'MyProp3'))

```

extractProtMoran

Moran Autocorrelation Descriptor

Description

Moran Autocorrelation Descriptor

Usage

```

extractProtMoran(
  x,
  props = c("CIDH920105", "BHAR880101", "CHAM820101", "CHAM820102", "CHOC760101",
            "BIGC670101", "CHAM810101", "DAYM780201"),
  nlag = 30L,
  customprops = NULL
)

```

Arguments

x	A character vector, as the input protein sequence.
props	A character vector, specifying the Accession Number of the target properties. 8 properties are used by default, as listed below: AccNo. CIDH920105 Normalized average hydrophobicity scales (Cid et al., 1992) AccNo. BHAR880101 Average flexibility indices (Bhaskaran-Ponnuswamy, 1988) AccNo. CHAM820101 Polarizability parameter (Charton-Charton, 1982) AccNo. CHAM820102 Free energy of solution in water, kcal/mole (Charton-Charton, 1982) AccNo. CHOC760101 Residue accessible surface area in tripeptide (Chothia, 1976) AccNo. BIGC670101 Residue volume (Bigelow, 1967) AccNo. CHAM810101 Steric parameter (Charton, 1981) AccNo. DAYM780201 Relative mutability (Dayhoff et al., 1978b)
nlag	Maximum value of the lag parameter. Default is 30.
customprops	A n x 21 named data frame contains n customize property. Each row contains one property. The column order for different amino acid types is 'AccNo', 'A', 'R', 'N', 'D', 'C', 'E', 'Q', 'G', 'H', 'I', 'L', 'K', 'M', 'F', 'P', 'S', 'T', 'W', 'Y', 'V', and the columns should also be <i>exactly</i> named like this. The AccNo column contains the properties' names. Then users should explicitly specify these properties with these names in the argument props. See the examples below for a demonstration. The default value for customprops is NULL.

extractProtMoreauBroto

Normalized Moreau-Broto Autocorrelation Descriptor

Description

Normalized Moreau-Broto Autocorrelation Descriptor

Usage

```
extractProtMoreauBroto(
  x,
  props = c("CIDH920105", "BHAR880101", "CHAM820101", "CHAM820102", "CHOC760101",
    "BIGC670101", "CHAM810101", "DAYM780201"),
  nlag = 30L,
  customprops = NULL
)
```

Arguments

x	A character vector, as the input protein sequence.
props	A character vector, specifying the Accession Number of the target properties. 8 properties are used by default, as listed below: AccNo. CIDH920105 Normalized average hydrophobicity scales (Cid et al., 1992) AccNo. BHAR880101 Average flexibility indices (Bhaskaran-Ponnuswamy, 1988) AccNo. CHAM820101 Polarizability parameter (Charton-Charton, 1982) AccNo. CHAM820102 Free energy of solution in water, kcal/mole (Charton-Charton, 1982) AccNo. CHOC760101 Residue accessible surface area in tripeptide (Chothia, 1976) AccNo. BIGC670101 Residue volume (Bigelow, 1967) AccNo. CHAM810101 Steric parameter (Charton, 1981) AccNo. DAYM780201 Relative mutability (Dayhoff et al., 1978b)
nlag	Maximum value of the lag parameter. Default is 30.
customprops	A n x 21 named data frame contains n customize property. Each row contains one property. The column order for different amino acid types is 'AccNo', 'A', 'R', 'N', 'D', 'C', 'E', 'Q', 'G', 'H', 'I', 'L', 'K', 'M', 'F', 'P', 'S', 'T', 'W', 'Y', 'V', and the columns should also be <i>exactly</i> named like this. The AccNo column contains the properties' names. Then users should explicitly specify these properties with these names in the argument props. See the examples below for a demonstration. The default value for customprops is NULL.

Details

This function calculates the normalized Moreau-Broto autocorrelation descriptor (Dim: length(props) * nlag).

Value

A length nlag named vector

References

AAindex: Amino acid index database. <https://www.genome.jp/dbget/aaindex.html>

Feng, Z.P. and Zhang, C.T. (2000) Prediction of membrane protein types based on the hydrophobic index of amino acids. *Journal of Protein Chemistry*, 19, 269-275.

Horne, D.S. (1988) Prediction of protein helix content from an autocorrelation analysis of sequence hydrophobicities. *Biopolymers*, 27, 451-477.

Sokal, R.R. and Thomson, B.A. (2006) Population structure inferred by local spatial autocorrelation: an usage from an Amerindian tribal population. *American Journal of Physical Anthropology*, 129, 121-131.

See Also

See [extractProtMoran](#) and [extractProtGeary](#) for Moran autocorrelation descriptors and Geary autocorrelation descriptors.

Examples

```
x = readFASTA(system.file('protseq/P00750.fasta', package = 'Rcpi'))[[1]]
extractProtMoreauBroto(x)
```

```
myprops = data.frame(AccNo = c("MyProp1", "MyProp2", "MyProp3"),
                      A = c(0.62, -0.5, 15), R = c(-2.53, 3, 101),
                      N = c(-0.78, 0.2, 58), D = c(-0.9, 3, 59),
                      C = c(0.29, -1, 47), E = c(-0.74, 3, 73),
                      Q = c(-0.85, 0.2, 72), G = c(0.48, 0, 1),
                      H = c(-0.4, -0.5, 82), I = c(1.38, -1.8, 57),
                      L = c(1.06, -1.8, 57), K = c(-1.5, 3, 73),
                      M = c(0.64, -1.3, 75), F = c(1.19, -2.5, 91),
                      P = c(0.12, 0, 42), S = c(-0.18, 0.3, 31),
                      T = c(-0.05, -0.4, 45), W = c(0.81, -3.4, 130),
                      Y = c(0.26, -2.3, 107), V = c(1.08, -1.5, 43))
```

```
# Use 4 properties in the AAindex database, and 3 customized properties
extractProtMoreauBroto(x, customprops = myprops,
                      props = c('CIDH920105', 'BHAR880101',
                                'CHAM820101', 'CHAM820102',
                                'MyProp1', 'MyProp2', 'MyProp3'))
```

extractProtPAAC

Pseudo Amino Acid Composition Descriptor

Description

Pseudo Amino Acid Composition Descriptor

Usage

```
extractProtPAAC(
  x,
  props = c("Hydrophobicity", "Hydrophilicity", "SideChainMass"),
  lambda = 30,
  w = 0.05,
  customprops = NULL
)
```

Arguments

x	A character vector, as the input protein sequence.
props	A character vector, specifying the properties used. 3 properties are used by default, as listed below: 'Hydrophobicity' Hydrophobicity value of the 20 amino acids 'Hydrophilicity' Hydrophilicity value of the 20 amino acids 'SideChainMass' Side-chain mass of the 20 amino acids
lambda	The lambda parameter for the PAAC descriptors, default is 30.
w	The weighting factor, default is 0.05.
customprops	A n x 21 named data frame contains n customize property. Each row contains one property. The column order for different amino acid types is 'AccNo', 'A', 'R', 'N', 'D', 'C', 'E', 'Q', 'G', 'H', 'I', 'L', 'K', 'M', 'F', 'P', 'S', 'T', 'W', 'Y', 'V', and the columns should also be <i>exactly</i> named like this. The AccNo column contains the properties' names. Then users should explicitly specify these properties with these names in the argument props. See the examples below for a demonstration. The default value for customprops is NULL.

Details

This function calculates the Pseudo Amino Acid Composition (PAAC) descriptor (Dim: $20 + \lambda$, default is 50).

Value

A length $20 + \lambda$ named vector

Note

Note the default 20×3 prop values have been already independently given in the function. Users could also specify other (up to 544) properties with the Accession Number in the [AAindex](#) data, with or without the default three properties, which means users should explicitly specify the properties to use.

References

Kuo-Chen Chou. Prediction of Protein Cellular Attributes Using Pseudo-Amino Acid Composition. *PROTEINS: Structure, Function, and Genetics*, 2001, 43: 246-255.

Type 1 pseudo amino acid composition. <http://www.csbio.sjtu.edu.cn/bioinf/PseAAC/type1.htm>

Kuo-Chen Chou. Using Amphiphilic Pseudo Amino Acid Composition to Predict Enzyme Sub-family Classes. *Bioinformatics*, 2005, 21, 10-19.

JACS, 1962, 84: 4240-4246. (C. Tanford). (The hydrophobicity data)

PNAS, 1981, 78:3824-3828 (T.P.Hopp & K.R.Woods). (The hydrophilicity data)

CRC Handbook of Chemistry and Physics, 66th ed., CRC Press, Boca Raton, Florida (1985). (The side-chain mass data)

R.M.C. Dawson, D.C. Elliott, W.H. Elliott, K.M. Jones, Data for Biochemical Research 3rd ed., Clarendon Press Oxford (1986). (The side-chain mass data)

See Also

See [extractProtAPAAC](#) for amphiphilic pseudo amino acid composition descriptor.

Examples

```
x = readFASTA(system.file('protseq/P00750.fasta', package = 'Rcpi'))[[1]]
extractProtPAAC(x)

myprops = data.frame(AccNo = c("MyProp1", "MyProp2", "MyProp3"),
                      A = c(0.62, -0.5, 15), R = c(-2.53, 3, 101),
                      N = c(-0.78, 0.2, 58), D = c(-0.9, 3, 59),
                      C = c(0.29, -1, 47), E = c(-0.74, 3, 73),
                      Q = c(-0.85, 0.2, 72), G = c(0.48, 0, 1),
                      H = c(-0.4, -0.5, 82), I = c(1.38, -1.8, 57),
                      L = c(1.06, -1.8, 57), K = c(-1.5, 3, 73),
                      M = c(0.64, -1.3, 75), F = c(1.19, -2.5, 91),
                      P = c(0.12, 0, 42), S = c(-0.18, 0.3, 31),
                      T = c(-0.05, -0.4, 45), W = c(0.81, -3.4, 130),
                      Y = c(0.26, -2.3, 107), V = c(1.08, -1.5, 43))

# Use 3 default properties, 4 properties in the AAindex database,
# and 3 customized properties
extractProtPAAC(x, customprops = myprops,
                props = c('Hydrophobicity', 'Hydrophilicity', 'SideChainMass',
                          'CIDH920105', 'BHAR880101',
                          'CHAM820101', 'CHAM820102',
                          'MyProp1', 'MyProp2', 'MyProp3'))
```

extractProtPSSM

Compute PSSM (Position-Specific Scoring Matrix) for given protein sequence

Description

Compute PSSM (Position-Specific Scoring Matrix) for given protein sequence

Usage

```
extractProtPSSM(
  seq,
  start.pos = 1L,
  end.pos = nchar(seq),
  psiblast.path = NULL,
```

```

makeblastdb.path = NULL,
database.path = NULL,
iter = 5,
silent = TRUE,
evaluate = 10L,
word.size = NULL,
gapopen = NULL,
gapextend = NULL,
matrix = "BLOSUM62",
threshold = NULL,
seg = "no",
soft.masking = FALSE,
culling.limit = NULL,
best.hit.overhang = NULL,
best.hit.score.edge = NULL,
xdrop.ungap = NULL,
xdrop.gap = NULL,
xdrop.gap.final = NULL,
window.size = NULL,
gap.trigger = 22L,
num.threads = 1L,
pseudocount = 0L,
inclusion.ethresh = 0.002
)

```

Arguments

seq	Character vector, as the input protein sequence.
start.pos	Optional integer denoting the start position of the fragment window. Default is 1, i.e. the first amino acid of the given sequence.
end.pos	Optional integer denoting the end position of the fragment window. Default is nchar(seq), i.e. the last amino acid of the given sequence.
psiblast.path	Character string indicating the path of the psiblast program. If NCBI Blast+ was previously installed in the operation system, the path will be automatically detected.
makeblastdb.path	Character string indicating the path of the makeblastdb program. If NCBI Blast+ was previously installed in the system, the path will be automatically detected.
database.path	Character string indicating the path of a reference database (a FASTA file).
iter	Number of iterations to perform for PSI-Blast.
silent	Logical. Whether the PSI-Blast running output should be shown or not (May not work on some Windows versions and PSI-Blast versions), default is TRUE.
evaluate	Expectation value (E) threshold for saving hits. Default is 10.
word.size	Word size for wordfinder algorithm. An integer ≥ 2 .
gapopen	Integer. Cost to open a gap.
gapextend	Integer. Cost to extend a gap.
matrix	Character string. The scoring matrix name (default is 'BLOSUM62').
threshold	Minimum word score such that the word is added to the BLAST lookup table. A real value ≥ 0 .

<code>seg</code>	Character string. Filter query sequence with SEG ('yes', 'window locut hicut', or 'no' to disable) Default is 'no'.
<code>soft.masking</code>	Logical. Apply filtering locations as soft masks? Default is FALSE.
<code>culling.limit</code>	An integer ≥ 0 . If the query range of a hit is enveloped by that of at least this many higher-scoring hits, delete the hit. Incompatible with <code>best.hit.overhang</code> and <code>best.hit.score.edge</code> .
<code>best.hit.overhang</code>	Best Hit algorithm overhang value (A real value ≥ 0 and ≤ 0.5 , recommended value: 0.1). Incompatible with <code>culling.limit</code> .
<code>best.hit.score.edge</code>	Best Hit algorithm score edge value (A real value ≥ 0 and ≤ 0.5 , recommended value: 0.1). Incompatible with <code>culling.limit</code> .
<code>xdrop.ungap</code>	X-dropoff value (in bits) for ungapped extensions.
<code>xdrop.gap</code>	X-dropoff value (in bits) for preliminary gapped extensions.
<code>xdrop.gap.final</code>	X-dropoff value (in bits) for final gapped alignment.
<code>window.size</code>	An integer ≥ 0 . Multiple hits window size, To specify 1-hit algorithm, use 0.
<code>gap.trigger</code>	Number of bits to trigger gapping. Default is 22.
<code>num.threads</code>	Integer. Number of threads (CPUs) to use in the BLAST search. Default is 1.
<code>pseudocount</code>	Integer. Pseudo-count value used when constructing PSSM. Default is 0.
<code>inclusion.ethresh</code>	E-value inclusion threshold for pairwise alignments. Default is 0.002.

Details

This function calculates the PSSM (Position-Specific Scoring Matrix) derived by PSI-Blast for given protein sequence or peptides. For given protein sequences or peptides, PSSM represents the log-likelihood of the substitution of the 20 types of amino acids at that position in the sequence. Note that the output value is not normalized.

Value

The original PSSM, a numeric matrix which has `end.pos - start.pos + 1` columns and 20 named rows.

Note

The function requires the `makeblastdb` and `psiblast` programs to be properly installed in the operation system or their paths provided.

The two command-line programs are included in the NCBI-BLAST+ software package. To install NCBI Blast+, just open the NCBI FTP site using web browser or FTP software: <ftp://anonymous@ftp.ncbi.nlm.nih.gov:21/blast/executables/blast+/LATEST/> then download the executable version of BLAST+ according to your operation system, and compile or install the downloaded source code or executable program.

Ubuntu/Debian users can directly use the command `sudo apt-get install ncbi-blast+` to install NCBI Blast+. For OS X users, download `ncbi-blast- ... .dmg` then install. For Windows users, download `ncbi-blast- ... .exe` then install.

References

Altschul, Stephen F., et al. "Gapped BLAST and PSI-BLAST: a new generation of protein database search programs." *Nucleic acids research* 25.17 (1997): 3389–3402.

Ye, Xugang, Guoli Wang, and Stephen F. Altschul. "An assessment of substitution scores for protein profile-profile comparison." *Bioinformatics* 27.24 (2011): 3356–3363.

Rangwala, Huzefa, and George Karypis. "Profile-based direct kernels for remote homology detection and fold recognition." *Bioinformatics* 21.23 (2005): 4239–4247.

See Also

[extractProtPSSMFeature](#) [extractProtPSSMAcc](#)

Examples

```
x = readFASTA(system.file('protseq/P00750.fasta', package = 'Rcpi'))[[1]]

dbpath = tempfile('tempdb', fileext = '.fasta')
invisible(file.copy(from = system.file('protseq/Plasminogen.fasta', package = 'Rcpi'), to = dbpath))
pssmmat = extractProtPSSM(seq = x, database.path = dbpath)
dim(pssmmat) # 20 x 562 (P00750: length 562, 20 Amino Acids)
```

extractProtPSSMAcc	<i>Profile-based protein representation derived by PSSM (Position-Specific Scoring Matrix) and auto cross covariance</i>
--------------------	--

Description

Profile-based protein representation derived by PSSM (Position-Specific Scoring Matrix) and auto cross covariance

Usage

```
extractProtPSSMAcc(pssmmat, lag)
```

Arguments

- pssmmat The PSSM computed by [extractProtPSSM](#).
- lag The lag parameter. Must be less than the number of amino acids in the sequence (i.e. the number of columns in the PSSM matrix).

Details

This function calculates the feature vector based on the PSSM by running PSI-Blast and auto cross covariance tranformation.

Value

A length $lag * 20^2$ named numeric vector, the element names are derived by the amino acid name abbreviation (crossed amino acid name abbreviation) and lag index.

References

Wold, S., Jonsson, J., Sjöström, M., Sandberg, M., & Rännar, S. (1993). DNA and peptide sequences and chemical processes multivariately modelled by principal component analysis and partial least-squares projections to latent structures. *Analytica chimica acta*, 277(2), 239–253.

See Also

[extractProtPSSM](#) [extractProtPSSMFeature](#)

Examples

```
x = readFASTA(system.file('protseq/P00750.fasta', package = 'Rcpi'))[[1]]

dbpath = tempfile('tempdb', fileext = '.fasta')
invisible(file.copy(from = system.file('protseq/Plasminogen.fasta', package = 'Rcpi'), to = dbpath))
pssmmat = extractProtPSSM(seq = x, database.path = dbpath)
pssmacc = extractProtPSSMAcc(pssmmat, lag = 3)
tail(pssmacc)
```

extractProtPSSMFeature

Profile-based protein representation derived by PSSM (Position-Specific Scoring Matrix)

Description

Profile-based protein representation derived by PSSM (Position-Specific Scoring Matrix)

Usage

```
extractProtPSSMFeature(pssmmat)
```

Arguments

pssmmat The PSSM computed by [extractProtPSSM](#).

Details

This function calculates the profile-based protein representation derived by PSSM. The feature vector is based on the PSSM computed by [extractProtPSSM](#). For a given sequence, The PSSM feature represents the log-likelihood of the substitution of the 20 types of amino acids at that position in the sequence. Each PSSM feature value in the vector represents the degree of conservation of a given amino acid type. The value is normalized to interval (0, 1) by the transformation $1/(1+e^{(-x)})$.

Value

A numeric vector which has $20 \times N$ named elements, where N is the size of the window (number of rows of the PSSM).

References

Ye, Xugang, Guoli Wang, and Stephen F. Altschul. "An assessment of substitution scores for protein profile-profile comparison." *Bioinformatics* 27.24 (2011): 3356–3363.

Rangwala, Huzefa, and George Karypis. "Profile-based direct kernels for remote homology detection and fold recognition." *Bioinformatics* 21.23 (2005): 4239–4247.

See Also

[extractProtPSSM](#) [extractProtPSSMAcc](#)

Examples

```
x = readFASTA(system.file('protseq/P00750.fasta', package = 'Rcpi'))[[1]]

dbpath = tempfile('tempdb', fileext = '.fasta')
invisible(file.copy(from = system.file('protseq/Plasminogen.fasta', package = 'Rcpi'), to = dbpath))
pssmmat = extractProtPSSM(seq = x, database.path = dbpath)
pssmfeature = extractProtPSSMFeature(pssmmat)
head(pssmfeature)
```

extractProtQSO	<i>Quasi-Sequence-Order (QSO) Descriptor</i>
----------------	--

Description

Quasi-Sequence-Order (QSO) Descriptor

Usage

```
extractProtQSO(x, nlag = 30, w = 0.1)
```

Arguments

- x A character vector, as the input protein sequence.
- nlag The maximum lag, default is 30.
- w The weighting factor, default is 0.1.

Details

This function calculates the Quasi-Sequence-Order (QSO) descriptor (Dim: $20 + 20 + (2 * nlag)$, default is 100).

Value

A length $20 + 20 + (2 * nlag)$ named vector

References

- Kuo-Chen Chou. Prediction of Protein Subcellar Locations by Incorporating Quasi-Sequence-Order Effect. *Biochemical and Biophysical Research Communications*, 2000, 278, 477-483.
- Kuo-Chen Chou and Yu-Dong Cai. Prediction of Protein Sucellular Locations by GO-FunD-PseAA Predictor. *Biochemical and Biophysical Research Communications*, 2004, 320, 1236-1239.
- Gisbert Schneider and Paul Wrede. The Rational Design of Amino Acid Sequences by Artificial Neural Networks and Simulated Molecular Evolution: Do Novo Design of an Idealized Leader Cleavage Site. *Biophys Journal*, 1994, 66, 335-344.

See Also

See [extractProtSOCN](#) for sequence-order-coupling numbers.

Examples

```
x = readFASTA(system.file('protseq/P00750.fasta', package = 'Rcpi'))[[1]]
extractProtQSO(x)
```

extractProtSOCN	<i>Sequence-Order-Coupling Numbers</i>
-----------------	--

Description

Sequence-Order-Coupling Numbers

Usage

```
extractProtSOCN(x, nlag = 30)
```

Arguments

x	A character vector, as the input protein sequence.
nlag	The maximum lag, default is 30.

Details

This function calculates the Sequence-Order-Coupling Numbers (Dim: nlag * 2, default is 60).

Value

A length nlag * 2 named vector

References

- Kuo-Chen Chou. Prediction of Protein Subcellar Locations by Incorporating Quasi-Sequence-Order Effect. *Biochemical and Biophysical Research Communications*, 2000, 278, 477-483.
- Kuo-Chen Chou and Yu-Dong Cai. Prediction of Protein Sucellular Locations by GO-FunD-PseAA Predictor. *Biochemical and Biophysical Research Communications*, 2004, 320, 1236-1239.
- Gisbert Schneider and Paul Wrede. The Rational Design of Amino Acid Sequences by Artificial Neural Networks and Simulated Molecular Evolution: Do Novo Design of an Idealized Leader Cleavage Site. *Biophys Journal*, 1994, 66, 335-344.

See Also

See [extractProtQSO](#) for quasi-sequence-order descriptors.

Examples

```
x = readFASTA(system.file('protseq/P00750.fasta', package = 'Rcpi'))[[1]]
extractProtSOCN(x)
```

extractProtTC

Tripeptide Composition Descriptor

Description

Tripeptide Composition Descriptor

Usage

```
extractProtTC(x)
```

Arguments

x A character vector, as the input protein sequence.

Details

This function calculates the Tripeptide Composition descriptor (Dim: 8000).

Value

A length 8000 named vector

References

M. Bhasin, G. P. S. Raghava. Classification of Nuclear Receptors Based on Amino Acid Composition and Dipeptide Composition. *Journal of Biological Chemistry*, 2004, 279, 23262.

See Also

See [extractProtAAC](#) and [extractProtDC](#) for Amino Acid Composition and Dipeptide Composition descriptors.

Examples

```
x = readFASTA(system.file('protseq/P00750.fasta', package = 'Rcpi'))[[1]]
extractProtTC(x)
```

getCPI

Generating Compound-Protein Interaction Descriptors

Description

Generating Compound-Protein Interaction Descriptors

Usage

```
getCPI(drugmat, protmat, type = c("combine", "tensorprod"))
```

Arguments

drugmat	The compound descriptor matrix.
protmat	The protein descriptor matrix.
type	The interaction type, one or two of "combine" and "tensorprod".

Details

This function calculates the compound-protein interaction descriptors by three types of interaction:

- combine - combine the two descriptor matrix, result has $(p1 + p2)$ columns
- tensorprod - calculate column-by-column (pseudo)-tensor product type interactions, result has $(p1 * p2)$ columns

Value

A matrix containing the compound-protein interaction descriptors

See Also

See [getPPI](#) for generating protein-protein interaction descriptors.

Examples

```
x = matrix(1:10, ncol = 2)
y = matrix(1:15, ncol = 3)

getCPI(x, y, 'combine')
getCPI(x, y, 'tensorprod')
getCPI(x, y, type = c('combine', 'tensorprod'))
getCPI(x, y, type = c('tensorprod', 'combine'))
```

getDrug	<i>Retrieve Drug Molecules in MOL and SMILES Format from Databases</i>
---------	--

Description

Retrieve Drug Molecules in MOL and SMILES Format from Databases

Usage

```
getDrug(  
  id,  
  from = c("pubchem", "chembl", "cas", "kegg", "drugbank"),  
  type = c("mol", "smile"),  
  parallel = 5  
)
```

Arguments

id	A character vector, as the drug ID(s).
from	The database, one of 'pubchem', 'chembl', 'cas', 'kegg', 'drugbank'.
type	The returned molecule format, mol or smile.
parallel	An integer, the parallel parameter, indicates how many process the user would like to use for retrieving the data (using RCurl), default is 5. For regular cases, we recommend a number less than 20.

Details

This function retrieves drug molecules in MOL and SMILES format from five databases.

Value

A length of id character vector, each element containing the corresponding drug molecule.

See Also

See [getProt](#) for retrieving protein sequences from three databases.

Examples

```
id = c('DB00859', 'DB00860')  
  
getDrug(id, 'drugbank', 'smile')
```

getFASTAFromKEGG	<i>Retrieve Protein Sequence in FASTA Format from the KEGG Database</i>
------------------	---

Description

Retrieve Protein Sequence in FASTA Format from the KEGG Database

Usage

```
getFASTAFromKEGG(id, parallel = 5)
```

Arguments

id	A character vector, as the protein ID.
parallel	An integer, the parallel parameter, indicates how many process the user would like to use for retrieving the data (using RCurl), default is 5. For regular cases, we recommend a number less than 20.

Details

This function retrieves protein sequences in FASTA format from the KEGG database.

Value

A list, each component contains one of the protein sequences in FASTA format.

See Also

See [getSeqFromKEGG](#) for retrieving protein represented by amino acid sequence from the KEGG database. See [readFASTA](#) for reading FASTA format files.

Examples

```
id = c('hsa:10161', 'hsa:10162')  
  
getFASTAFromKEGG(id)
```

getFASTAFromUniProt	<i>Retrieve Protein Sequence in FASTA Format from the UniProt Database</i>
---------------------	--

Description

Retrieve Protein Sequence in FASTA Format from the UniProt Database

Usage

```
getFASTAFromUniProt(id, parallel = 5)
```

Arguments

<code>id</code>	A character vector, as the protein ID.
<code>parallel</code>	An integer, the parallel parameter, indicates how many process the user would like to use for retrieving the data (using RCurl), default is 5. For regular cases, we recommend a number less than 20.

Details

This function retrieves protein sequences in FASTA format from the UniProt database.

Value

A list, each component contains one of the protein sequences in FASTA format.

References

UniProt. <https://www.uniprot.org/>

UniProt REST API Documentation. <https://www.uniprot.org/help/api>

See Also

See [getSeqFromUniProt](#) for retrieving protein represented by amino acid sequence from the UniProt database. See [readFASTA](#) for reading FASTA format files.

Examples

```
id = c('P00750', 'P00751', 'P00752')  
getFASTAFromUniProt(id)
```

`getMolFromCAS`*Retrieve Drug Molecules in InChI Format from the CAS Database*

Description

Retrieve Drug Molecules in InChI Format from the CAS Database

Usage

```
getMolFromCAS(id, parallel = 5)
```

Arguments

<code>id</code>	A character vector, as the CAS drug ID.
<code>parallel</code>	An integer, the parallel parameter, indicates how many process the user would like to use for retrieving the data (using RCurl), default is 5. For regular cases, we recommend a number less than 20.

Details

This function retrieves drug molecules in InChI format from the CAS database. CAS database only provides InChI data, so here we return the molecule in InChI format, users could convert them to SMILES format using Open Babel or other third-party tools.

Value

A length of id character vector, each element containing the corresponding drug molecule.

See Also

See [getDrug](#) for retrieving drug molecules in MOL and SMILES Format from other databases.

Examples

```
id = '52-67-5' # Penicillamine  
  
getMolFromCAS(id)
```

getMolFromChEMBL	<i>Retrieve Drug Molecules in MOL Format from the ChEMBL Database</i>
------------------	---

Description

Retrieve Drug Molecules in MOL Format from the ChEMBL Database

Usage

```
getMolFromChEMBL(id, parallel = 5)
```

Arguments

id	A character vector, as the ChEMBL drug ID.
parallel	An integer, the parallel parameter, indicates how many process the user would like to use for retrieving the data (using RCurl), default is 5. For regular cases, we recommend a number less than 20.

Details

This function retrieves drug molecules in MOL format from the ChEMBL database.

Value

A length of id character vector, each element containing the corresponding drug molecule.

See Also

See [getSmiFromChEMBL](#) for retrieving drug molecules in SMILES format from the ChEMBL database.

Examples

```
id = 'CHEMBL1430' # Penicillamine  
  
getMolFromChEMBL(id)
```

getMolFromDrugBank	<i>Retrieve Drug Molecules in MOL Format from the DrugBank Database</i>
--------------------	---

Description

Retrieve Drug Molecules in MOL Format from the DrugBank Database

Usage

```
getMolFromDrugBank(id, parallel = 5)
```

Arguments

id	A character vector, as the DrugBank drug ID.
parallel	An integer, the parallel parameter, indicates how many process the user would like to use for retrieving the data (using RCurl), default is 5. For regular cases, we recommend a number less than 20.

Details

This function retrieves drug molecules in MOL format from the DrugBank database.

Value

A length of id character vector, each element containing the corresponding drug molecule.

See Also

See [getSmiFromDrugBank](#) for retrieving drug molecules in SMILES format from the DrugBank database.

Examples

```
id = 'DB00859' # Penicillamine  
  
getMolFromDrugBank(id)
```

getMolFromKEGG	<i>Retrieve Drug Molecules in MOL Format from the KEGG Database</i>
----------------	---

Description

Retrieve Drug Molecules in MOL Format from the KEGG Database

Usage

```
getMolFromKEGG(id, parallel = 5)
```

Arguments

id	A character vector, as the KEGG drug ID.
parallel	An integer, the parallel parameter, indicates how many process the user would like to use for retrieving the data (using RCurl), default is 5. For regular cases, we recommend a number less than 20.

Details

This function retrieves drug molecules in MOL format from the KEGG database.

Value

A length of id character vector, each element containing the corresponding drug molecule.

See Also

See [getSmiFromKEGG](#) for retrieving drug molecules in SMILES format from the KEGG database.

Examples

```
id = 'D00496' # Penicillamine  
getMolFromKEGG(id)
```

getMolFromPubChem	<i>Retrieve Drug Molecules in MOL Format from the PubChem Database</i>
-------------------	--

Description

Retrieve Drug Molecules in MOL Format from the PubChem Database

Usage

```
getMolFromPubChem(id, parallel = 5)
```

Arguments

id	A character vector, as the PubChem drug ID.
parallel	An integer, the parallel parameter, indicates how many processes the user would like to use for retrieving the data (using RCurl), default is 5. For regular cases, we recommend a number less than 20.

Details

This function retrieves drug molecules in MOL format from the PubChem database.

Value

A length of id character vector, each element containing the corresponding drug molecule.

See Also

See [getSmiFromPubChem](#) for retrieving drug molecules in SMILES format from the PubChem database.

Examples

```
id = c('7847562', '7847563') # Penicillamine  
getMolFromPubChem(id)
```

getPDBFromRCSBPDB	<i>Retrieve Protein Sequence in PDB Format from RCSB PDB</i>
-------------------	--

Description

Retrieve Protein Sequence in PDB Format from RCSB PDB

Usage

```
getPDBFromRCSBPDB(id, parallel = 5)
```

Arguments

id	A character vector, as the protein ID.
parallel	An integer, the parallel parameter, indicates how many process the user would like to use for retrieving the data (using RCurl), default is 5. For regular cases, we recommend a number less than 20.

Details

This function retrieves protein sequences in PDB format from RCSB PDB.

Value

A list, each component contains one of the protein sequences in PDB format.

See Also

See [getSeqFromRCSBPDB](#) for retrieving protein represented by amino acid sequence from the RCSB PDB database.

Examples

```
id = c('4HHB', '4FF9')  
getPDBFromRCSBPDB(id)
```

getPPI*Generating Protein-Protein Interaction Descriptors*

Description

Generating Protein-Protein Interaction Descriptors

Usage

```
getPPI(protmat1, protmat2, type = c("combine", "tensorprod", "entrywise"))
```

Arguments

protmat1	The first protein descriptor matrix, must have the same ncol with protmat2.
protmat2	The second protein descriptor matrix, must have the same ncol with protmat1.
type	The interaction type, one or more of "combine", "tensorprod", and "entrywise".

Details

This function calculates the protein-protein interaction descriptors by three types of interaction:

- combine - combine the two descriptor matrix, result has $(p + p)$ columns
- tensorprod - calculate column-by-column (pseudo)-tensor product type interactions, result has $(p * p)$ columns
- entrywise - calculate entrywise product and entrywise sum of the two matrices, then combine them, result has $(p + p)$ columns

Value

A matrix containing the protein-protein interaction descriptors

See Also

See [getCPI](#) for generating compound-protein interaction descriptors.

Examples

```
x = matrix(1:10, ncol = 2)
y = matrix(5:14, ncol = 2)

getPPI(x, y, type = 'combine')
getPPI(x, y, type = 'tensorprod')
getPPI(x, y, type = 'entrywise')
getPPI(x, y, type = c('combine', 'tensorprod'))
getPPI(x, y, type = c('combine', 'entrywise'))
getPPI(x, y, type = c('entrywise', 'tensorprod'))
getPPI(x, y, type = c('combine', 'entrywise', 'tensorprod'))
```

`getProt`*Retrieve Protein Sequence in various Formats from Databases*

Description

Retrieve Protein Sequence in various Formats from Databases

Usage

```
getProt(  
  id,  
  from = c("uniprot", "kegg", "pdb"),  
  type = c("fasta", "pdb", "aaseq"),  
  parallel = 5  
)
```

Arguments

<code>id</code>	A character vector, as the protein ID(s).
<code>from</code>	The database, one of 'uniprot', 'kegg', or 'pdb'.
<code>type</code>	The returned protein format, one of fasta, pdb, or aaseq.
<code>parallel</code>	An integer, the parallel parameter, indicates how many process the user would like to use for retrieving the data (using RCurl), default is 5. For regular cases, we recommend a number less than 20.

Details

This function retrieves protein sequence in various formats from three databases.

Value

A length of `id` character list, each element containing the corresponding protein sequence(s) or file(s).

See Also

See [getDrug](#) for retrieving drug molecules from five databases.

Examples

```
id = c('P00750', 'P00751', 'P00752')  
  
getProt(id, from = 'uniprot', type = 'aaseq')
```

`getSeqFromKEGG`*Retrieve Protein Sequence from the KEGG Database*

Description

Retrieve Protein Sequence from the KEGG Database

Usage

```
getSeqFromKEGG(id, parallel = 5)
```

Arguments

<code>id</code>	A character vector, as the protein ID.
<code>parallel</code>	An integer, the parallel parameter, indicates how many process the user would like to use for retrieving the data (using RCurl), default is 5. For regular cases, we recommend a number less than 20.

Details

This function retrieves protein represented by amino acid sequence from the KEGG database.

Value

A list, each component contains one of the protein represented by amino acid sequence(s).

See Also

See [getFASTAFromKEGG](#) for retrieving protein sequence in FASTA format from the KEGG database.

Examples

```
id = c('hsa:10161', 'hsa:10162')  
  
getSeqFromKEGG(id)
```

`getSeqFromRCSBPDB`*Retrieve Protein Sequence from RCSB PDB*

Description

Retrieve Protein Sequence from RCSB PDB

Usage

```
getSeqFromRCSBPDB(id, parallel = 5)
```

Arguments

id	A character vector, as the protein ID.
parallel	An integer, the parallel parameter, indicates how many process the user would like to use for retrieving the data (using RCurl), default is 5. For regular cases, we recommend a number less than 20.

Details

This function retrieves protein sequences from RCSB PDB.

Value

A list, each component contains one of the protein represented by amino acid sequence(s).

See Also

See [getPDBFromRCSBPDB](#) for retrieving protein in PDB format from the RCSB PDB database.

Examples

```
id = c('4HHB', '4FF9')  
getSeqFromRCSBPDB(id)
```

getSeqFromUniProt	<i>Retrieve Protein Sequence from the UniProt Database</i>
-------------------	--

Description

Retrieve Protein Sequence from the UniProt Database

Usage

```
getSeqFromUniProt(id, parallel = 5)
```

Arguments

id	A character vector, as the protein ID.
parallel	An integer, the parallel parameter, indicates how many process the user would like to use for retrieving the data (using RCurl), default is 5. For regular cases, we recommend a number less than 20.

Details

This function retrieves protein represented by amino acid sequence from the UniProt database.

Value

A list, each component contains one of the protein represented by amino acid sequence(s).

References

UniProt. <https://www.uniprot.org/>

UniProt REST API Documentation. <https://www.uniprot.org/help/api>

See Also

See [getFASTAFromUniProt](#) for retrieving protein sequences in FASTA format from the UniProt database.

Examples

```
id = c('P00750', 'P00751', 'P00752')
```

```
getSeqFromUniProt(id)
```

getSmiFromChEMBL	<i>Retrieve Drug Molecules in SMILES Format from the ChEMBL Database</i>
------------------	--

Description

Retrieve Drug Molecules in SMILES Format from the ChEMBL Database

Usage

```
getSmiFromChEMBL(id, parallel = 5)
```

Arguments

id	A character vector, as the ChEMBL drug ID.
parallel	An integer, the parallel parameter, indicates how many process the user would like to use for retrieving the data (using RCurl), default is 5. For regular cases, we recommend a number less than 20.

Details

This function retrieves drug molecules in SMILES format from the ChEMBL database.

Value

A length of id character vector, each element containing the corresponding drug molecule.

See Also

See [getMolFromChEMBL](#) for retrieving drug molecules in MOL format from the ChEMBL database.

Examples

```
id = 'CHEMBL1430' # Penicillamine
```

```
getSmiFromChEMBL(id)
```

getSmiFromDrugBank	<i>Retrieve Drug Molecules in SMILES Format from the DrugBank Database</i>
--------------------	--

Description

Retrieve Drug Molecules in SMILES Format from the DrugBank Database

Usage

```
getSmiFromDrugBank(id, parallel = 5)
```

Arguments

id	A character vector, as the DrugBank drug ID.
parallel	An integer, the parallel parameter, indicates how many process the user would like to use for retrieving the data (using RCurl), default is 5. For regular cases, we recommend a number less than 20.

Details

This function retrieves drug molecules in SMILES format from the DrugBank database.

Value

A length of id character vector, each element containing the corresponding drug molecule.

See Also

See [getMolFromDrugBank](#) for retrieving drug molecules in MOL format from the DrugBank database.

Examples

```
id = 'DB00859' # Penicillamine  
getSmiFromDrugBank(id)
```

getSmiFromKEGG	<i>Retrieve Drug Molecules in SMILES Format from the KEGG Database</i>
----------------	--

Description

Retrieve Drug Molecules in SMILES Format from the KEGG Database

Usage

```
getSmiFromKEGG(id, parallel = 5)
```

Arguments

id	A character vector, as the KEGG drug ID.
parallel	An integer, the parallel parameter, indicates how many process the user would like to use for retrieving the data (using RCurl), default is 5. For regular cases, we recommend a number less than 20.

Details

This function retrieves drug molecules in SMILES format from the KEGG database.

Value

A length of id character vector, each element containing the corresponding drug molecule.

See Also

See [getMolFromKEGG](#) for retrieving drug molecules in MOL format from the KEGG database.

Examples

```
id = 'D00496' # Penicillamine  
  
getSmiFromKEGG(id)
```

getSmiFromPubChem	<i>Retrieve Drug Molecules in SMILES Format from the PubChem Database</i>
-------------------	---

Description

Retrieve Drug Molecules in SMILES Format from the PubChem Database

Usage

```
getSmiFromPubChem(id, parallel = 5)
```

Arguments

id	A character vector, as the PubChem drug ID.
parallel	An integer, the parallel parameter, indicates how many processes the user would like to use for retrieving the data (using RCurl), default is 5. For regular cases, we recommend a number less than 20.

Details

This function retrieves drug molecules in SMILES format from the PubChem database.

Value

A length of id character vector, each element containing the corresponding drug molecule.

See Also

See [getMolFromPubChem](#) for retrieving drug molecules in MOL format from the PubChem database.

Examples

```
id = c('7847562', '7847563') # Penicillamine

getSmiFromPubChem(id)
```

OptAA3d	<i>OptAA3d.sdf - 20 Amino Acids Optimized with MOE 2011.10 (Semiempirical AM1)</i>
---------	--

Description

OptAA3d.sdf - 20 Amino Acids Optimized with MOE 2011.10 (Semiempirical AM1)

Details

OptAA3d.sdf - 20 Amino Acids Optimized with MOE 2011.10 (Semiempirical AM1)

Value

OptAA3d data

Examples

```
# This example requires the rcdk package
# library('rcdk')
# optaa3d = load.molecules(system.file('sysdata/OptAA3d.sdf', package = 'Rcpi'))
# view.molecule.2d(optaa3d[[1]]) # view the first amino acid
```

readFASTA	<i>Read Protein Sequences in FASTA Format</i>
-----------	---

Description

Reads protein sequences in FASTA format.

Usage

```
readFASTA(
  file = system.file("protseq/P00750.fasta", package = "Rcpi"),
  legacy.mode = TRUE,
  seqonly = FALSE
)
```

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 P00750.fasta file which is present in the protseq directory of the Rcp package.
legacy.mode	If set to TRUE, lines starting with a semicolon ';' are ignored. Default value is TRUE.
seqonly	If set to 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). Default value is FALSE.

Value

Character vector.

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.

See Also

See [readPDB](#) for reading protein sequences in PDB format.

Examples

```
P00750 = readFASTA(system.file('protseq/P00750.fasta', package = 'Rcp'))
P00750
```

readMolFromSDF	<i>Read Molecules from SDF Files and Return Parsed Java Molecular Object</i>
----------------	--

Description

Read Molecules from SDF Files and Return Parsed Java Molecular Object

Usage

```
readMolFromSDF(sdf file)
```

Arguments

sdf file	Character vector, containing SDF file location(s).
----------	--

Details

This function reads molecules from SDF files and return parsed Java molecular object needed by extractDrug... functions.

Value

A list, containing parsed Java molecular object.

See Also

See [readMolFromSmi](#) for reading molecules by SMILES string and returning parsed Java molecular object.

Examples

```
sdf = system.file('compseq/DB00859.sdf', package = 'Rcpi')
sdfs = c(system.file('compseq/DB00859.sdf', package = 'Rcpi'),
          system.file('compseq/DB00860.sdf', package = 'Rcpi'))

mol = readMolFromSDF(sdf)
mols = readMolFromSDF(sdfs)
```

readMolFromSmi	<i>Read Molecules from SMILES Files and Return Parsed Java Molecular Object or Plain Text List</i>
----------------	--

Description

Read Molecules from SMILES Files and Return Parsed Java Molecular Object or Plain Text List

Usage

```
readMolFromSmi(smifile, type = c("mol", "text"))
```

Arguments

smifile	Character vector, containing SMILES file location(s).
type	'mol' or 'text'. 'mol' returns parsed Java molecular object, used for 'text' returns (plain-text) character string list. For common molecular descriptors and fingerprints, use 'mol'. For descriptors and fingerprints calculated by OpenBabel, i.e. functions named extractDrugOB...(), use 'text'.

Details

This function reads molecules from SMILES strings and return parsed Java molecular object or plain text list needed by extractDrug...() functions.

Value

A list, containing parsed Java molecular object or character strings.

See Also

See [readMolFromSDF](#) for reading molecules from SDF files and returning parsed Java molecular object.

Examples

```
smi = system.file('vignettesdata/FDAMDD.smi', package = 'Rcpi')

mol1 = readMolFromSmi(smi, type = 'mol')
mol2 = readMolFromSmi(smi, type = 'text')
```

readPDB

Read Protein Sequences in PDB Format

Description

Read Protein Sequences in PDB Format

Usage

```
readPDB(file = system.file("protseq/4HHB.pdb", package = "Rcpi"))
```

Arguments

file	The name of the file which the sequences in PDB 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 4HHB.PDB file which is present in the protseq directory of the Rcpi package.
------	---

Details

This function reads protein sequences in PDB (Protein Data Bank) format, and return the amino acid sequences represented by single-letter code.

Value

A character vector, representing the amino acid sequence of the single-letter code.

References

Protein Data Bank Contents Guide: Atomic Coordinate Entry Format Description, Version 3.30. Accessed 2013-06-26. https://files.wwpdb.org/pub/pdb/doc/format_descriptions/Format_v33_Letter.pdf

See Also

See [readFASTA](#) for reading protein sequences in FASTA format.

Examples

```
Seq4HHB = readPDB(system.file('protseq/4HHB.pdb', package = 'Rcpi'))
Seq4HHB
```

searchDrug

*Parallelized Drug Molecule Similarity Search by Molecular Fingerprints Similarity or Maximum Common Substructure Search***Description**

Parallelized Drug Molecule Similarity Search by Molecular Fingerprints Similarity or Maximum Common Substructure Search

Usage

```
searchDrug(
  mol,
  molddb,
  cores = 2,
  method = c("fp", "mcs"),
  fptype = c("standard", "extended", "graph", "hybrid", "maccs", "estate", "pubchem",
    "kr", "shortestpath", "fp2", "fp3", "fp4", "obmaccs"),
  fpsim = c("tanimoto", "euclidean", "cosine", "dice", "hamming"),
  mcssim = c("tanimoto", "overlap"),
  ...
)
```

Arguments

mol	The query molecule. The location of a sdf file containing one molecule.
molddb	The molecule database. The location of a sdf file containing all the molecules to be searched with.
cores	Integer. The number of CPU cores to use for parallel search, default is 2. Users could use the detectCores() function in the parallel package to see how many cores they could use.
method	'fp' or 'mcs'. Search by molecular fingerprints or by maximum common substructure searching.
fptype	The fingerprint type, only available when method = 'fp'. Rcpis supports 13 types of fingerprints, including 'standard', 'extended', 'graph', 'hybrid', 'maccs', 'estate', 'pubchem', 'kr', 'shortestpath', 'fp2', 'fp3', 'fp4', 'obmaccs'.
fpsim	Similarity measure type for fingerprint, only available when method = 'fp'. Including 'tanimoto', 'euclidean', 'cosine', 'dice' and 'hamming'. See calcDrugFPSim for details.
mcssim	Similarity measure type for maximum common substructure search, only available when method = 'mcs'. Including 'tanimoto' and 'overlap'.
...	Other possible parameter for maximum common substructure search, see calcDrugMCSSim for available options.

Details

This function does compound similarity search derived by various molecular fingerprints with various similarity measures or derived by maximum common substructure search. This function runs for a query compound against a set of molecules.

Value

Named numerical vector. With the decreasing similarity value of the molecules in the database.

Examples

```
mol = system.file('compseq/DB00530.sdf', package = 'Rcpi')
# DrugBank ID DB00530: Erlotinib
molddb = system.file('compseq/tyrphostin.sdf', package = 'Rcpi')
# Database composed by searching 'tyrphostin' in PubChem and filtered by Lipinski's Rule of Five

searchDrug(mol, molddb, cores = 4, method = 'fp', fptype = 'maccs', fpsim = 'hamming')
searchDrug(mol, molddb, cores = 4, method = 'fp', fptype = 'fp2', fpsim = 'tanimoto')
searchDrug(mol, molddb, cores = 4, method = 'mcs', mcssim = 'tanimoto')
```

segProt

Protein Sequence Segmentation

Description

Protein Sequence Segmentation

Usage

```
segProt(
  x,
  aa = c("A", "R", "N", "D", "C", "E", "Q", "G", "H", "I", "L", "K", "M", "F", "P", "S",
        "T", "W", "Y", "V"),
  k = 7
)
```

Arguments

x	A character vector, as the input protein sequence.
aa	A character, the amino acid type. one of 'A', 'R', 'N', 'D', 'C', 'E', 'Q', 'G', 'H', 'I', 'L', 'K', 'M', 'F', 'P', 'S', 'T', 'W', 'Y', 'V'.
k	A positive integer, specifies the window size (half of the window), default is 7.

Details

This function extracts the segmentations from the protein sequence.

Value

A named list, each component contains one of the segmentations (a character string), names of the list components are the positions of the specified amino acid in the sequence.

Examples

```
x = readFASTA(system.file('protseq/P00750.fasta', package = 'Rcpi'))[[1]]
segProt(x, aa = 'R', k = 5)
```

Index

* internal

Rcpi-package, 5

AA2DACOR, 6

AA3DMORSE, 6

AAACF, 7

AABLOSUM100, 7

AABLOSUM45, 8

AABLOSUM50, 8

AABLOSUM62, 9

AABLOSUM80, 9

AABurden, 10

AAConn, 10

AAConst, 11

AACPSA, 11

AADescAll, 12

AAEdgeAdj, 12

AAEigIdx, 13

AAFGC, 13

AAGeom, 14

AAGETAWAY, 14

AAindex, 15, 111, 122

AAInfo, 15

AAMetaInfo, 16

AAMOE2D, 16

AAMOE3D, 17

AAMolProp, 17

AAPAM120, 18

AAPAM250, 18

AAPAM30, 19

AAPAM40, 19

AAPAM70, 20

AARandic, 20

AARDF, 21

AATopo, 21

AATopoChg, 22

AAWalk, 22

AAWHIM, 23

acc, 23

calcDrugFPSim, 24

calcDrugMCSSim, 25

calcParProtGOSim, 27, 29, 30

calcParProtSeqSim, 28, 28, 30, 31

calcTwoProtGOSim, 28, 29, 31

calcTwoProtSeqSim, 30

checkProt, 31

convMolFormat, 32

extractDrugAIO, 37

extractDrugALOGP, 38

extractDrugAminoAcidCount, 39

extractDrugApol, 39

extractDrugAromaticAtomsCount, 40

extractDrugAromaticBondsCount, 41

extractDrugAtomCount, 42

extractDrugAutocorrelationCharge, 42

extractDrugAutocorrelationMass, 43

extractDrugAutocorrelationPolarizability, 44

extractDrugBCUT, 45

extractDrugBondCount, 46

extractDrugBPol, 47

extractDrugCarbonTypes, 48

extractDrugChiChain, 49

extractDrugChiCluster, 50

extractDrugChiPath, 51

extractDrugChiPathCluster, 52

extractDrugCPSA, 53

extractDrugDescOB, 54

extractDrugECI, 55

extractDrugEstate, 56, 57

extractDrugEstateComplete, 57, 57

extractDrugExtended, 58, 59

extractDrugExtendedComplete, 58, 59

extractDrugFMF, 60

extractDrugFragmentComplexity, 61

extractDrugGraph, 62, 63

extractDrugGraphComplete, 62, 63

extractDrugGravitationalIndex, 64

extractDrugHBondAcceptorCount, 65

extractDrugHBondDonorCount, 66

extractDrugHybridization, 66, 68

extractDrugHybridizationComplete, 67, 67

extractDrugHybridizationRatio, 68

extractDrugIPMolecularLearning, 69

extractDrugKappaShapeIndices, 70

- extractDrugKierHallSmarts, [71](#)
- extractDrugKR, [73](#), [74](#)
- extractDrugKRComplete, [74](#), [74](#)
- extractDrugLargestChain, [75](#)
- extractDrugLargestPiSystem, [76](#)
- extractDrugLengthOverBreadth, [76](#)
- extractDrugLongestAliphaticChain, [77](#)
- extractDrugMACCS, [78](#), [79](#)
- extractDrugMACCSComplete, [78](#), [79](#)
- extractDrugMannholdLogP, [79](#)
- extractDrugMDE, [80](#)
- extractDrugMomentOfInertia, [81](#)
- extractDrugOBFP2, [82](#)
- extractDrugOBFP3, [83](#)
- extractDrugOBFP4, [84](#)
- extractDrugOBMACCS, [85](#)
- extractDrugPetitjeanNumber, [86](#)
- extractDrugPetitjeanShapeIndex, [87](#)
- extractDrugPubChem, [88](#), [89](#)
- extractDrugPubChemComplete, [88](#), [88](#)
- extractDrugRotatableBondsCount, [89](#)
- extractDrugRuleOfFive, [90](#)
- extractDrugShortestPath, [91](#), [92](#)
- extractDrugShortestPathComplete, [91](#), [92](#)
- extractDrugStandard, [93](#), [94](#)
- extractDrugStandardComplete, [93](#), [94](#)
- extractDrugTPSA, [95](#)
- extractDrugVABC, [96](#)
- extractDrugVAdjMa, [96](#)
- extractDrugWeight, [97](#)
- extractDrugWeightedPath, [98](#)
- extractDrugWHIM, [99](#)
- extractDrugWienerNumbers, [100](#)
- extractDrugXLogP, [101](#)
- extractDrugZagrebIndex, [102](#)
- extractPCMBLOSUM, [103](#)
- extractPCMDescScales, [24](#), [104](#), [109](#)
- extractPCMFAScales, [105](#)
- extractPCMDSScales, [106](#)
- extractPCMPPropScales, [24](#), [107](#), [109](#)
- extractPCMScales, [24](#), [105](#), [107](#), [108](#), [108](#)
- extractProtAAC, [109](#), [116](#), [130](#)
- extractProtAPAAC, [110](#), [123](#)
- extractProtCTDC, [112](#), [113](#), [114](#)
- extractProtCTDD, [112](#), [113](#), [114](#)
- extractProtCTDT, [112](#), [113](#), [114](#)
- extractProtCTriad, [115](#)
- extractProtDC, [110](#), [115](#), [130](#)
- extractProtGeary, [116](#), [119](#), [121](#)
- extractProtMoran, [117](#), [118](#), [121](#)
- extractProtMoreauBroto, [117](#), [119](#), [120](#)
- extractProtPAAC, [111](#), [121](#)
- extractProtPSSM, [123](#), [126–128](#)
- extractProtPSSMAcc, [126](#), [126](#), [128](#)
- extractProtPSSMFeature, [126](#), [127](#), [127](#)
- extractProtQSO, [128](#), [130](#)
- extractProtSOCN, [129](#), [129](#)
- extractProtTC, [110](#), [116](#), [130](#)
- getCPI, [131](#), [139](#)
- getDrug, [132](#), [135](#), [140](#)
- getFASTAFromKEGG, [133](#), [141](#)
- getFASTAFromUniProt, [133](#), [143](#)
- getMolFromCAS, [134](#)
- getMolFromChEMBL, [135](#), [143](#)
- getMolFromDrugBank, [136](#), [144](#)
- getMolFromKEGG, [136](#), [145](#)
- getMolFromPubChem, [137](#), [146](#)
- getPDBFromRCSBPDB, [138](#), [142](#)
- getPPI, [131](#), [139](#)
- getProt, [132](#), [140](#)
- getSeqFromKEGG, [133](#), [141](#)
- getSeqFromRCSBPDB, [138](#), [141](#)
- getSeqFromUniProt, [134](#), [142](#)
- getSmiFromChEMBL, [135](#), [143](#)
- getSmiFromDrugBank, [136](#), [144](#)
- getSmiFromKEGG, [137](#), [144](#)
- getSmiFromPubChem, [138](#), [145](#)
- getwd, [147](#), [149](#)
- OptAA3d, [11](#), [17](#), [146](#)
- Rcpi (Rcpi-package), [5](#)
- Rcpi-package, [5](#)
- readFASTA, [133](#), [134](#), [146](#), [149](#)
- readMolFromSDF, [147](#), [148](#)
- readMolFromSmi, [148](#), [148](#)
- readPDB, [147](#), [149](#)
- searchDrug, [150](#)
- segProt, [151](#)