Package ‘veriNA3d’

December 12, 2018

Type Package

Title Biological Structure Analysis

Version 0.99.0

Date 2018-12-12

Author Diego Gallego

Maintainer Diego Gallego `<diego.gallego@irbbarcelona.org>`

Description Structural bioinformatics is a hard field we work to make easier.
We provide tools to extract information from PDB structures, manage these data and save the results in different kinds of plots. The package was developed (and so the examples and vignettes) to do data mining with PDB data focused on RNA, but most of its functionalities are extensible to the analysis of any PDB structure.

License GPL-3

Depends R (>= 3.5)

Imports methods, graphics, grDevices, stats, utils, bio3d, circlize, parallel, jsonlite, plot3D, MASS, RColorBrewer, RANN

LazyData true

RoxygenNote 6.1.0

Roxygen list(markdown = TRUE)

Encoding UTF-8

Suggests knitr, rmarkdown

VignetteBuilder knitr

NeedsCompilation no

R topics documented:

`applyToPDB` *Applies a function over a list of PDB ID entries.*

Description

Given a function of interest, it is applied to all the PDB entries. See supported functions in ?queryFunctions.

Usage

```
applyToPDB(FUNCTION, listpdb = NULL, as.df = TRUE, cores = 1,
  progressbar = TRUE, ...)
```

Arguments

`FUNCTION`	A function of interest.
`listpdb`	A list/vector containing the PDB IDs of interest. If NULL, the complete list of PDB entries is downloaded and used.
`as.df`	A logical that stands for "as.data.frame". If TRUE, the output will be returned in the form of a data.frame, otherwise a list.

`cores`	Number of CPU cores to be used.
`progressbar`	A logical to print in screen a progress bar.
`...`	optional arguments to FUNCTION.

Value

A data.frame with the PDB IDs (first colunm) and the output of the function of interest (second column) or a list with the results.

Author(s)

Diego Gallego

Examples

```
listpdb <- c("1s72", "1bau", "1rna")
applyToPDB(queryTechnique, listpdb)
```

`checkNuc` *Check nucleotides*

Description

From a nucleic acid structure (pdb object), it checks the presence of all nucleotide atoms, bond distances, chain breaks and others.

Usage

```
checkNuc(pdb, model = 1, chain = "all", id = NULL, refatm = "C4'",
  force = FALSE)
```

Arguments

`pdb`	A pdb object as obtained from cifAsPDB or read.cif/read.pdb (bio3d package).
`model`	A string with the desired model number.
`chain`	A string with the desired chain id.
`id`	A string with the ID of the input pdb structure.
`refatm`	A string with the atom to use to identify the nucelotides. Important to analyse models with just (in example) phosphate atoms, in which refatm should be set to "P" (it was thought when analysing the structure with PDB code: 1Y1Y).
`force`	A logical to force the analysis. Useful when the function does not recognise a nucleic acid in the structure (e.g. because all bases are non-canonical: 1PBL, 1XV6, 1DV4 ...)

Value

A data.frame with the data for every nucleotide.

Author(s)

Diego Gallego

Examples

```
data <- checkNuc(cifAsPDB("1am0"))
broken <- which(data$Break == TRUE)
data[broken, ] ## See the places in which the chain is broken
```

`CIF-class` *An S4 class to represent a structure parsed from its mmCIF file.*

Description

All mmCIF files in the PDB at date 2018-Feb-19th contain (just) 14 common attributes, which are represented in the CIF objects herein with the same names as found in mmCIF documentation `http://mmcif.wwpdb.org/`.

Slots

`entry`	The ID code
`audit_conform`	'mmCIF' dictionary version
`database_2`	Cross-reference ID codes to other databases
`pdbx_database_status`	Deposition data
`audit_author`	Authors
`entity`	Entities (molecules & ions) in the structure
`chem_comp`	Residues (ATOM & HETATM) in the structure
`exptl`	Experimental technique
`struct`	Author description of the structure
`struct_keywords`	Author description key words
`struct_asym`	Chain-entity equivalences
`atom_sites`	Details about the crystallographic cell
`atom_type`	Details about the atoms in structure
`atom_site`	The atomic coordinates

Author(s)

Diego Gallego

See Also

To create CIF objects use `cifParser()`

`cif_accessors` *Accessors to a CIF object*

Description

S4 method to access the contents of CIF objects.

Usage

```
cifEntry(x)

cifAudit_conform(x)

cifDatabase_2(x)

cifPdbx_database_status(x)

cifAudit_author(x)

cifEntity(x)

cifChem_comp(x)

cifExptl(x)

cifStruct(x)

cifStruct_keywords(x)

cifStruct_asym(x)

cifAtom_sites(x)

cifAtom_type(x)

cifAtom_site(x)

## S4 method for signature 'CIF'
cifEntry(x)

## S4 method for signature 'CIF'
cifAudit_conform(x)

## S4 method for signature 'CIF'
cifDatabase_2(x)

## S4 method for signature 'CIF'
cifPdbx_database_status(x)

## S4 method for signature 'CIF'
cifAudit_author(x)
```

```
## S4 method for signature 'CIF'
cifEntity(x)

## S4 method for signature 'CIF'
cifChem_comp(x)

## S4 method for signature 'CIF'
cifExptl(x)

## S4 method for signature 'CIF'
cifStruct(x)

## S4 method for signature 'CIF'
cifStruct_keywords(x)

## S4 method for signature 'CIF'
cifStruct_asym(x)

## S4 method for signature 'CIF'
cifAtom_sites(x)

## S4 method for signature 'CIF'
cifAtom_type(x)

## S4 method for signature 'CIF'
cifAtom_site(x)
```

Arguments

`x`	a CIF object

Value

```
* {cifEntry} `Character` with the mmCIF PDB ID
* {cifAudit_conform} `Character` vector with dictionary version
* {cifDatabase_2} `Data.frame` with cross-references
* {cifPdbx_database_status} `Character` vector with deposition data
* {cifAudit_author} `Data.frame` with author names
* {cifEntity} `Data.frame` with molecules & ions in the structure
* {cifChem_comp} `Data.frame` with residues records in the structure
* {cifExptl} `Character` vector with experimental technique
* {cifStruct} `Character` vector with author description of the
    structure
* {cifStruct_keywords} `Character` vector with author selected key words
* {cifStruct_asym} `Data.frame` with chain-entity equivalences
* {cifAtom_sites} `Character` vector with details about the
    crystallographic cell
* {cifAtom_type} `Data.frame` with about the atoms in structure
* {cifAtom_site} `Data.frame` with atomic coordinates
```

Author(s)

Diego Gallego

Examples

```
cif <- cifParser("1bau")
coordinates <- cifAtom_site(cif)
```

---

cifAsPDB — *Coerce CIF S4 object to pdb S3 object as found in bio3d package*

Description

Coerces CIF to pdb class

Usage

```
cifAsPDB(cif, model = NULL, chain = NULL, alt = c("A"),
  verbose = FALSE)

## S4 method for signature 'CIF'
cifAsPDB(cif, model = NULL, chain = NULL,
  alt = c("A"), verbose = FALSE)

## S4 method for signature 'character'
cifAsPDB(cif, model = NULL, chain = NULL,
  alt = c("A"), verbose = FALSE)
```

Arguments

`cif`	A CIF object as obtained from cifParser. It can also accept a 4-character PDB ID.
`model`	A string with the model number (in case you are only interested in a particular model) If NULL, all models are parsed and can be selected afterwards using `selectModel`. By default, the "atom" attribute of the output will contain only the first model.
`chain`	A string with the chain identifier (in case you are only interested in a particular chain). If NULL, all chains are included.
`alt`	A string or a vector of strings with the desired alternative records.
`verbose`	A logical indicating whether to print details of the process.

Value

A pdb object compatible with bio3d (Grant et al. 2006) functions.

Author(s)

Diego Gallego

Examples

```
cif <- cifParser("1bau")
pdb <- cifAsPDB(cif)
```

---

cifParser *Parse coordinates from CIF files*

Description

Given a file or PDB ID, the function parses the coordinates of the structure. It can also read all the fields of the mmCIF format.

Usage

```
cifParser(pdbID, verbose = FALSE)

## S4 method for signature 'ANY'
cifParser(pdbID, verbose = FALSE)
```

Arguments

`pdbID`	A 4-character string that matches a structure in the Protein Data Bank (or an existing file in disk).
`verbose`	A logical indicating whether to print details of the process.

Value

A S4 CIF object

Author(s)

Diego Gallego

Examples

```
cif <- cifParser("1bau")
```

`classifyNA` *Classify RNA or DNA structures*

Description

The functions classify a structure in the following groups:

- NoRNA or NoDNA: the structure does not contain RNA or it is shorter than a threshold set by "length".
- nakedRNA or nakedDNA: the only molecule(s) in the structure is RNA or DNA.
- protRNA or protDNA: the PDB contains a protein-nuc complex.
- DprotRNA or DprotDNA: the PDB contains a protein-nuc complex and the protein has D aminoacids.
- DNARNA: the PDB contains DNA-RNA.
- PNARNA or PNADNA: the PDB contains PNA-RNA.
- ANARNA: the PDB contains ANA-RNA.
- LNARNA: the PDB contains LNA-RNA.
- ligandRNA or ligandDNA: the RNA/DNA is interacting with an organic ligand. Ions are not considered as ligands in this class.

Usage

```
classifyRNA(pdbID, length = 3, force = FALSE, ...)

classifyDNA(pdbID, force = FALSE, ...)
```

Arguments

`pdbID`	A 4-character string that matches a structure ID in the Protein Data Bank.
`length`	A positive integer. Minimum numer of nucleotides to consider RNA as a polymer. An RNA shorter than this threshold is classified in the NoRNA group.
`force`	A logical to force the query instead of getting presaved data.
`...`	Arguments to be passed to query function (see ?queryFunctions).

Details

In `classifyRNA`, nucleic acid hybrids are considered RNA, while in in `classifyDNA` they are considered DNA (e.g. pdb ID 2HVR).

Value

A string with the type of RNA.

Author(s)

Diego Gallego

Examples

```
classifyRNA("1S72")
```

cleanByPucker *Subset nucleotide data according with puckering*

Description

Function to clean raw data after the pipeline `pipeNucData()`. It takes a data.frame that should have the columns "pu_phase", "delta" and "Dp", and returns the nucleotides matching the desired puckering state.

Usage

```
cleanByPucker(ntinfo, surenorth = FALSE, suresouth = FALSE,
  pucker = "C3'endo", range = NULL, verbose = TRUE)
```

Arguments

`ntinfo`	A data.frame. The output of `pipeNucData()`.
`surenorth`	A logical to return nucleotides in north with restrictions in delta and Dp.
`suresouth`	A logical to return nucleotides in south with restrictions in delta and Dp.
`pucker`	A string with the puckering state of interest. Only necessary if surenorth and suresouth are FALSE and range is NULL. When using this option, only the phase is used to subset the data.
`range`	A numeric vector with the desired phaserange. Only used if no other argument above could be applyed.
`verbose`	A logical to print details of the process.

Value

An integer vector with the nucleotides (ntID) matching the desired puckering state.

Author(s)

Diego Gallego

Examples

```
    ntinfo <- pipeNucData("1bau")
    north_ntID <- cleanByPucker(ntinfo, surenorth=TRUE)
    north <- ntinfo[ntinfo$ntID %in% north_ntID,]
```

countEntities *Count entities*

Description

For a given pdbID, the function gets the Entity data and counts the number of instances of the different entities (e.g. the number of different RNA, the number of different proteins...).

Usage

```
countEntities(pdbID, force = FALSE, ...)
```

Arguments

pdbID	A 4-character string that matches a structure ID in the Protein Data Bank.
force	A logical to force the query instead of getting presaved data.
...	Arguments to be passed to query function (see ?queryFunctions).

Value

A list with the number of instances of each entity.

Author(s)

Diego Gallego

Examples

```
countEntities("1S72")
```

dssr *Dissecting the Spatial Structure of RNA with DSSR*

Description

Wrapper function to execute DSSR (see reference below) on a DNA or RNA structure and parse the result.

Usage

```
dssr(pdb, exefile = "x3dna-dssr", dssrargs = c("--nmr", "--torsion360",
  "--more"), verbose = FALSE)
```

Arguments

`pdb`	It can be: • A 4 character string corresponding to a PDB ID • A pdb/mmcif file • A pdb object as provided by `cifAsPDB()` or `bio3d::read.pdb()`.
`exefile`	A string with the program name
`dssrargs`	A vector of strings with the desired arguments to feed DSSR
`verbose`	A logical indicating whether to print details of the process.

Value

A list with the json output of DSSR

Author(s)

Diego Gallego

References

```
Lu, X.J. et al. (2015). "DSSR: an integrated software tool for
dissecting the spatial structure of RNA." Nucleic Acids Res.
43(21), e142
```

Examples

```
# Not run
# dssr_1bau <- dssr("1bau")
```

entities — *List of NA containing PDB IDs and related data*

Description

Number of each type of entity in a set of Nucleic Acid containing structures.

Usage

```
data(entities)
```

Format

An object of class data.frame

pdbID: PDB ID.

RNA: Number of different RNA entities

DNA: Number of different DNA entities

Hybrid: Number of different Hybrid DNA/RNA entities

PNA: Number of different PNA entities

Prot: Number of different Protein (L) entities

Dprot: Number of different Protein (D) entities

Ligands: Number of different ligands

Water: It's 1 when there's water in the structure and 0 when it is not

Other: Number of different "other" entities

Value

data.frame with a list and features of NA PDB IDs

---

eRMSD — *Compute the epsilon RMSD between two RNA structures*

Description

Given two RNA with the same length, the functions calculates its epsilon RMSD, as defined by Bottaro et al. 2014 (The role of nucleobase interactions in RNA structure and dynamics), reproducing baRNAba software. Methods allow as input CIF S4 objects `cifParser()`, pdb S3 objects (cifAsPDB/read.pdb/read.cif) or matrices containing the "r" vectors of the desired structures.

Usage

```
eRMSD(cif1 = NULL, cif2 = NULL, pdb1 = NULL, pdb2 = NULL,
  rvectors1 = NULL, rvectors2 = NULL)

## S4 method for signature 'CIF,CIF'
eRMSD(cif1 = NULL, cif2 = NULL)

## S4 method for signature 'ANY,ANY'
eRMSD(pdb1 = NULL, pdb2 = NULL, rvectors1 = NULL,
  rvectors2 = NULL)
```

Arguments

`cif1`	A CIF object as otained from `cifParser()`.
`cif2`	A CIF object as otained from `cifParser()`.
`pdb1`	A pdb object as obtained from cifAsPDB or read.cif/read.pdb (from bio3d package).
`pdb2`	A pdb object as obtained from cifAsPDB or read.cif/read.pdb (from bio3d package).
`rvectors1`	A data.frame as obtained from `rVector()` using simple_out=TRUE.
`rvectors2`	A data.frame as obtained from `rVector()` using simple_out=TRUE.

Value

A numeric with the epsilon RMSD between the two structures

Author(s)

Diego Gallego

Examples

```
cif <- cifParser("2d18")
model1 <- selectModel(cif=cif, model=1)
model3 <- selectModel(cif=cif, model=3)
eRMSD <- eRMSD(cif1=model1, cif2=model3)
```

`fastquery` *List of NA containing PDB IDs and related data*

Description

Presaved data to speed up some queries.

Usage

```
data(fastquery)
```

Format

An object of class data.frame with the following fields:

pdbID: PDB ID.

Technique: Experimental technique.

Resol: Resolution. For NMR structures it contains an empty string.

DNAclass: Output of classifyDNA function

RNAclassOver0: Output of classifyRNA with length=0 or length=1. If the structure has one or two nucleotides, it is also considered an RNA containing structure.

RNAclassOver2: Output of classifyRNA with length=3. RNA molecules shorter than 3 are classified as NoRNA.

Value

data.frame with a list and features of NA PDB IDs

`findBindingSite` *Function to get data about the atoms in interacting site.*

Description

For pdb structures with protein-nucleic acid complexes, the function finds the atoms in the interacting site. It allows the user to set as reference the nucleic acid, the protein, or particular desired chains.

Usage

```
findBindingSite(pdb, cutoff = 5, select = "Nuc", nchain = NULL,
  pchain = NULL, hydrogens = FALSE, byres = FALSE, verbose = FALSE,
  ...)
```

Arguments

`pdb`	A cif/pdb object obtained from cifParser/read.pdb respectively or a pdb ID so that the function can download the data.
`cutoff`	A numeric to set the maximum distance for atoms to be returned.
`select`	A string that should match "Nuc", "Prot", "DNA" or "RNA", to be used as reference.
`nchain`	A string with the nucleic acid chain to get data about. If NULL, all of them are selected (according with select argument).
`pchain`	A string with the protein chain to get data about. If NULL, all of them are selected.
`hydrogens`	A logical to use the hydrogens in the structure or remove them.
`byres`	A logical to indicate if the output should be referred to the residues rather than atoms.
`verbose`	A logical to print details of the process.
`...`	Arguments to selectModel and/or alt records.

Value

A data.frame with the atomic distances in the interacting site.

Author(s)

Diego Gallego

Examples

```
pdb <- cifParser("1b3t") # A protein-DNA complex
data <- findBindingSite(pdb, select="DNA", byres=TRUE)
```

`getAltRepres` *Get Alternative Representants*

Description

This function is closely related with getLeontisList(). From its output, the family members of each equivalence class are checked for desired features. The first member that matches all the desired features is returned.

Usage

```
getAltRepres(rnalist, technique = NULL, resol = NULL, type = NULL,
  length = 3, progressbar = TRUE, verbose = FALSE)
```

Arguments

`rnalist`	The output of getLeontisList.
`technique`	One or more techniques of interest (For correct use, see example below). For the list of techniques, see "veriNA3d:::.allowedtechs".
`resol`	A positive real number to specify a desired resolution.
`type`	A string indicating the type of desired RNA, according with the classifyRNA function.
`length`	To be passed to classifyRNA.
`progressbar`	A logical to print in screen a progress bar.
`verbose`	A logical to print details of the process.

Value

A data.frame with info about all the "Equivalence Classes" and the selected Representants according with the specified conditions.

Author(s)

Diego Gallego

Examples

```
rnalist <- getLeontisList(release=3.2, threshold="1.5A")
alternative <- getAltRepres(rnalist=rnalist,
                            type="nakedRNA")
```

`getLeontisList` *Download Representative List of RNA structures*

Description

According with Leontis & Zirbel, 2012 (Nonredundant 3D Structure Datasets for RNA Knowledge Extraction and Benchmarking), the PDB contains structures that are redundant. Their work resulted in the BGSU RNA Site (http://rna.bgsu.edu/rna3dhub/nrlist/) where one can find weekly releases of Representative Sets of RNA structures (formerly called non-redundant lists). This function access their website and returns the desired list.

Usage

```
getLeontisList(release = "current", threshold = "all")
```

Arguments

`release`	A number indicating the list of interest.
`threshold`	A string that matches one of the lists in the BGSU RNA site ("1.5A", "2.0A", "2.5A", "3.0A", "3.5A", "4.0A", "20.0A", "all"). Note that "all" returns structures solved by any technique.

Value

A data frame with the list of Equivalence Classes and the Representant and Members of each Equivalence Class. Note that the output is formated according with Leontis&Zirbel nomenclature (AAAA|M|C), where "AAAA" is the PDB accession code, "M" is the model and "C" is the Chain to be used.

Author(s)

Diego Gallego

Examples

```
    data <- getLeontisList(release=3.2, threshold="1.5A")
```

---

hasHetAtm *Check if a given PDB contains the ligand/modbase of interest*

Description

Given a 4-character string (PDB ID) and a ligand/modbase ID, the function checks the presence of the ligand/modres in the given PDB ID. To check for the presence of sodium ions use hetAtms="Na" instead of NA. If you are interested on the whole list of heterogeneous atoms see `queryHetAtms()`.

Usage

```
hasHetAtm(pdbID, hetAtms)
```

Arguments

`pdbID`	A 4-character string.
`hetAtms`	A string with the ligand/modbase ID.

Value

A logical. TRUE if the given hetAtms is present in the structure.

Author(s)

Diego Gallego

Examples

```
hasHetAtm("1s72", "MG") # Check if structure has Magnesium ion
```

`measureElenoDist` *Computes distances between the atoms of interest in a mmCIF structure*

Description

Given a pdb object (or a pdb ID), the function computes the distances between the desired atoms and returns the closest ones. Note that eleno numbers might be different in the PDB vs mmCIF formats and this may lead to errors.

Usage

```
measureElenoDist(pdb, model = NULL, refeleno, eleno, n = 1,
  cutoff = c(0, 5), verbose = FALSE, detailedoutput = TRUE,
  data_of_interest = NULL)
```

Arguments

`pdb`	A pdb object obtained as from `cifAsPDB()` or read.pdb/read.cif (bio3d functions)
`model`	The model of interest to use in the calculations. The first model is always the default.
`refeleno`	A vector of eleno (element number) to take as reference.
`eleno`	A vector of eleno to measure the distances.
`n`	An integer indicating how many closests atoms to return. The default n=1 returns only the closest atom; n=2 would return the two closest atoms and so on. If NULL, any number of atoms within the cutoff will be returned.
`cutoff`	A numeric vector indicating the distance range to consider in angstroms. Atoms further than the cutoff won't be returned.
`verbose`	A logical indicating whether to print details of the process.
`detailedoutput`	A logical indicating whether to include additional information for each atom (see data_of_interest below). If FALSE, only the eleno (element number) and distances are returned.
`data_of_interest`	A vector of strings. Only used if detailedoutput is TRUE. The vector should only contain the strings between the following: "type", "elety", "alt", "resid", "chain", "resno", "insert", "x", "y", "z", "o", "b", "entid", "elesy", "charge", "asym_id", "seq_id", "comp_id", "atom_id", "model". The selected fields will be returned for both atoms.

Value

A data.frame with the nearest atom neighbours information. Fields suffixed with '_A' refer to the atoms used as reference. Fields suffixed with '_B' refer to the 'contacting'/closest atoms.

Author(s)

Diego Gallego

Examples

```
## Dowload cif file and save coordinates data
cif <- cifParser("1enn")
coordinates <- cifAtom_site(cif)

## Find atom numbers for desired entities (e.g. water and DNA)
water_eleno <- coordinates[coordinates$label_atom_id == "O", "id"]
dna_eleno <- coordinates[coordinates$label_comp_id %in%
                                    c("DA", "DT", "DG", "DU"), "id"]

## Find which DNA atoms are in 5 Angstroms distance from the water
data <- measureElenoDist(cif, refeleno=water_eleno, eleno=dna_eleno,
                              n=NULL, cutoff=5)

## To see the data
head(data)
```

`measureEntityDist` *Computes distances between all the atoms of selected entities in a mmCIF structure*

Description

Given a cif object (or a pdb ID), the function computes the distances between atoms of the selected entity IDs. For each atom/residue of the reference entity the function returns the closest atoms of the other entities. This function is a wrapper of `measureElenoDist()` and simplifies its use. If you are unfamiliar with the concept of entities in a mmCIF structure see example below.

Usage

```
measureEntityDist(cif, model = NULL, refent, entities = c("all"), ...)
```

Arguments

`cif`	A cif object obtained from cifParser or a pdb ID.
`model`	The model of interest to use in the calculations. The first model is always the default.
`refent`	A string with the entity ID of reference. The distance output will be referred to the atoms/residues of this entity.
`entities`	A character vector with the entities of interest. The default "all" will select all of them except the refent.
`...`	Additional arguments to be passed to `measureElenoDist()`

Value

A data.frame with the nearest atoms neighbour information.

Author(s)

Diego Gallego

Examples

```
## To see the entities of a given structure use:
cif <- cifParser("1enn")
cifEntity(cif)

## Supose you are interested on the interactions of water and DNA
water_entity <- 5
dna_entity <- 1

## Find which DNA atoms are in 5 Angstroms distance from the water
data <- measureEntityDist(cif, refent=water_entity,
            entities=dna_entity, n=10, cutoff=5)
## An equivalent run without downloading the cif file previously
data <- measureEntityDist("1enn", refent=water_entity,
            entities=dna_entity, n=10, cutoff=5)

## This option is better than using the example in ?measureElenoDist,
## since this way it would also take into account modified residues, if
## any.
```

`measureNuc` *Obtain desired nucleotide measurements*

Description

From a nucleic acid structure (pdb object), it computes the desired atomic distances, angles, dihedral angles, puckering conformation and Dp distance (See definition of Dp in MolProbity paper by Chen et al. 2010).

Usage

```
measureNuc(pdb, model = 1, chain = "all", v_shifted = TRUE,
  b_shifted = TRUE, distances = "default", angles = "default",
  torsionals = "default", pucker = TRUE, Dp = TRUE, refatm = "C4'",
  force = FALSE)
```

Arguments

`pdb`	A pdb object as obtained from cifAsPDB or read.cif/read.pdb (bio3d package).
`model`	A string with the desired model number.
`chain`	A string with the desired chain id.
`v_shifted`	A logical. If TRUE, puckering angles (nu0 to nu4) are returned in the range 0 to 360 degrees. Otherwise, -180 to +180.
`b_shifted`	A logical. If TRUE, backbone angles, chi and kappa are returned in the range 0 to 360 degrees. Otherwise, -180 to +180.
`distances`	A data.frame indicating all the intra and inter-nucleotide atomic distances of interest. See details section. A default option is preconfigured to simplify the use of the function and can be seen typing 'veriNA3d::.distances'.

`angles`	A data.frame indicating all the intra and inter-nucleotide angles of interest. See details section. A default option is preconfigured to simplify the use of the function and can be seen typing 'veriNA3d::.angles'.
`torsionals`	A data.frame indicating all the intra and inter- nucleotide torsional angles of interest. See details section. A default option is preconfigured to simplify the use of the function and can be seen typing 'veriNA3d::.torsionals'.
`pucker`	A logical indicating whether to compute the puckering.
`Dp`	A logical indicating whether to compute the Dp distance.
`refatm`	A string with the atom to use to identify the nucelotides. Important to analyse models with just (in example) phosphate atoms, in which refatm should be set to "P" (it was thought when analysing the structure with PDB code: 1Y1Y).
`force`	A logical to force the analysis. Useful when the function does not recognise a nucleic acid in the structure (e.g. because all bases are non-canonical: 1PBL, 1XV6, 1DV4 ...)

Details

The format of 'distances', 'angles' and 'torsionals' is: First column should indicate the first atom, second column second atom (and so on in the case of angles and torsional angles). An extra last column is optional and should contain the names to identify each measurement in the output. Plane atom names are interpreted as intra- nucleotide measurments. For inter-nucleotide measurments use the prefix "pre_" or "post_" before the atom name. In example, to compute all inter-phosphate distances, use as argument:
distances=data.frame(atomA=c("P"), atomB=c("post_P"), labels=c("interphosphate"), stringsAsFactors=FALSE)

Value

A data.frame with the measurements for every nucleotide.

Author(s)

Diego Gallego

Examples

```
distances <- data.frame(atomA=c("P"), atomB=c("post_P"),
                        labels=c("interphosphate"),
                        stringsAsFactors=FALSE)
measureNuc(cifAsPDB("1bna"), distances=distances, angles=NULL,
           torsionals=NULL, Dp=NULL)
```

pipeNucData *Obtain nucleotide details from a data set of RNA structures*

Description

Pipeline to generate a data.frame with the desired info for a list of PDB. Nucleotides are labeled with a unique identifier (column ntID).

Usage

```
pipeNucData(pdbID, model = NULL, chain = NULL, range = c(3, 1e+05),
  path = NULL, extension = NULL, cores = 1, progressbar = TRUE,
  ...)
```

Arguments

`pdbID`	A list/vector containing the desired PDB IDs or a list of pdb objects as provided by "read.pdb", "read.cif", "cifParser" ...
`model`	A vector with same length of pdbID containing the desired model for each pdbID. If all models are desired, use "all". If no models are specified, the first one will be used for each pdbID.
`chain`	A vector with same length of pdbID containing the desired chain for each pdbID. If no chain is specified, all chains will be analysed by default. Non-nucleic acid chains will be ignored.
`range`	A numeric vector with two values to indicate the desired length range for the Nucleic Acid chains. If a chain falls outside the range, it is not analysed.
`path`	Directory in which the PDB/CIF files can be found (if NULL, the function will download them). If you provide a "path", make sure the file names are the PDB IDs followed by ".cif" or "pdb". The function will find them using the strings in pdbID, so make sure you use the same case.
`extension`	A string matching the files extension (e.g. ".pdb", ".cif", "pdb.gz", "cif.gz"). Only necessary if the PDB files are to be read from disk and a path is provided.
`cores`	Number of CPU cores to be used.
`progressbar`	A logical to print in screen a progress bar.
`...`	Arguments to be passed to `measureNuc()`

Value

A data.frame with data about every nucleotide in the input set

Author(s)

Diego Gallego

Examples

```
## This is a toy example, see vignettes for real-world usages.
pdblist <- list("1bau", "2rn1")
model <- list("1", "2")
chain <- list("all", "all")
ntinfo <- pipeNucData(pdbID=pdblist, model=model, chain=chain)
```

`pipeProtNucData` *Obtain nucleotide-protein interactions from a data set of structures*

Description

Pipeline to generate a data.frame with the data about the closests nucleotides to the protein for a list of PDB. The data can be related to unique nucleotide indentifiers (ntID) by providing the output of the independent pipeline `pipeNucData()`.

Usage

```
pipeProtNucData(pdbID, model = NULL, chain = NULL, ntinfo = NULL,
  path = NULL, extension = NULL, cores = 1, progressbar = TRUE,
  cutoff = 15, ...)
```

Arguments

`pdbID`	A list/vector containing the desired PDB IDs or a list of pdb objects as provided by "read.pdb", "read.cif", "cifParser" ...
`model`	A vector with same length of pdbID containing the desired model for each pdbID. If all models are desired, use "all". If no models are specified, the first one will be used for each pdbID
`chain`	A vector with same length of pdbID containing the desired chain for each pdbID. If no chain is specified, all chains will be analysed by default. Non-nucleic acid chains will be ignored.
`ntinfo`	Optional. A data.frame obtained from `pipeNucData()` for the same dataset (or bigger), but not smaller.
`path`	Directory in which the PDB/CIF files can be found (if NULL, the function will download them). If you provide a "path", make sure the file names are the PDB IDs followed by ".cif" or "pdb". The function will find them using the strings in pdbID, so make sure you use the same case.
`extension`	A string matching the files extension (e.g. ".pdb", ".cif", "pdb.gz", "cif.gz"). Only necessary if the PDB files are to be read from disk and a path is provided.
`cores`	Number of CPU cores to be used.
`progressbar`	A logical to print in screen a progress bar.
`cutoff`	A numeric with the maximum distance to return. To be passed to `findBindingSite()`
`...`	Additional arguments to be passed to `findBindingSite()`

Value

A data.frame with data about the atomic distances in the interacting sites of every structure in the input set.

Author(s)

Diego Gallego

Examples

```
## This is a toy example, see vignettes for more usages.
pdblist <- list("1nyb", "2ms1")
aantinfo <- pipeProtNucData(pdbID=pdblist)
```

`plotCategorical` *Barplot wrapper*

Description

Function to make more straigtforward the process of ploting a barplot for categorical data.

Usage

```
plotCategorical(ntinfo, field, ntID = NULL, na.rm = FALSE,
  main = NULL, cex = 0.5, file = NULL, width = 15, height = 15,
  bg = "white", units = "cm", res = 200)
```

Arguments

`ntinfo`	A data.frame with the input data. It should contain the columns with the desired categorical data and a column labeled ntID.
`field`	The column name with the desired data.
`ntID`	A vector of integers with the desired nucleotides of analysis. If NULL all the nucleotides in the data.frame will be used.
`na.rm`	A logical to remove missing data.
`main`	A string with the title of the plot.
`cex`	To be passed to the par() function
`file`	A string with the name of the output file. If NULL, the plot will be printed to screen.
`width`	The width of the plot (passed to the png() function)
`height`	The height of the plot (passed to the png() function)
`bg`	The background color of the plot (passed to the png() function)
`units`	The unit to measure height and width (passed to the png() function)
`res`	Resolution (passed to the png() function)

Value

A barplot with the categorical data of interest, which can be directly saved to a ".png" file.

Author(s)

Diego Gallego

Examples

```
## To see all the types of trinucleotides in the dataset:
ntinfo <- pipeNucData("1bau")
plotCategorical(ntinfo=ntinfo, field="localenv")
```

---

plotCircularDistribution

Plot a scatter&frequency circular plot for angular data

Description

For a vector of angular data (0 to 360), the function plots the distribution in a circular format.

Usage

```
plotCircularDistribution(data, clockwise = FALSE, start.degree = 0,
  main = NULL)
```

Arguments

`data`	A numeric vector with the data to plot.
`clockwise`	A logical indicating the sense in which the data should be displayed.
`start.degree`	An integer with the position in which the data starts being ploted.
`main`	A string with the title of the plot.

Value

A circular plot with the input data

Author(s)

Diego Gallego

Examples

```
ntinfo <- pipeNucData("1bau")
C3endo_ntID <- cleanByPucker(ntinfo, pucker="C3'endo")
C3endo <- ntinfo[ntinfo$ntID %in% C3endo_ntID,]
plotCircularDistribution(C3endo[, "delta"])
```

plotEtaTheta *Plot eta-theta*

Description

Function to plot eta-theta and highlight the most populated regions.

Usage

```
plotEtaTheta(ntinfo, ntID = NULL, dens = NULL, bandwidths = NULL,
  eta = "eta", theta = "theta", drawcontour = TRUE,
  sd_over_mean_contours = c(1, 2, 4), highlight_helical = TRUE,
  points = NULL, colpoints = "red", file = NULL, width = 15,
  height = 15, bg = "white", units = "cm", res = 200)

plotEtaTheta3D(ntinfo, ntID = NULL, dens = NULL, bandwidths = NULL,
  eta = "eta", theta = "theta", defaultview = NULL, thetaplot,
  phiplot, cleanerview = FALSE, file = NULL, width = 15,
  height = 15, bg = "white", units = "cm", res = 600)
```

Arguments

`ntinfo`	A data.frame with the input data. It should contain the columns with eta and theta data and a column labeled ntID.
`ntID`	A vector of integers with the desired nucleotides of analysis. If NULL all the nucleotides in the data.frame will be used.
`dens`	The output of a kernel density estimation (e.g. kde2d function) over the data of interest. If it is NULL and drawcontour=TRUE, it will be computed under the hood.
`bandwidths`	In case dens=NULL and drawcontour=TRUE, it will be passed to kde2d().
`eta`	A string with the parameter to be placed in the x axis.
`theta`	A string with the parameter to be placed in the y axis.
`drawcontour`	A logical to highlight the most populated regions of the plot.
`sd_over_mean_contours`	A numeric vector with the standard deviations over the mean to plot the contours (in case drawcontour=TRUE).
`highlight_helical`	A logical to highlight the helical region of the eta-theta plot.
`points`	An integer vector for advanced usage of the function. It should contain the ntID of the nucleotides to print in a different color.
`colpoints`	A string with the desired color if points is not NULL.
`file`	A string with the name of the output file. If NULL, the plot will be printed to screen.
`width`	The width of the plot (passed to the png() function)
`height`	The height of the plot (passed to the png() function)
`bg`	The background color of the plot (passed to the png() function)
`units`	The unit to measure height and width (passed to the png() function)

`res`	Resolution (passed to the png() function)
`defaultview`	A string to set different default options. Chose between ’2Dupview’, ’leftview’ and ’rightview’.
`thetaplot`	Argument to be pased to `persp3D()` to set the view perspective.
`phiplot`	Argument to be pased to `persp3D()` to est the view perspective.
`cleanerview`	A logical to remove the lower density region to get a cleaner plot.

Value

A plot in screen, which can be directly saved to a ".png" file. * plotEtaTheta A scatter plot with eta-theta values. * plotEtaTheta3D A density map of the data in 3D.

Author(s)

Diego Gallego

Examples

```
ntinfo <- pipeNucData("1bau")
C3endo_ntID <- cleanByPucker(ntinfo, pucker="C3'endo")
plotEtaTheta(ntinfo=ntinfo, ntID=C3endo_ntID)
```

`plotSetOfDistributions` *Plot distribution of desired angles in circular plots*

Description

Given a data.frame with nucleotides data, it generates a series of circular plots for the desired angles. NA in the data are ignored.

Usage

```
plotSetOfDistributions(ntinfo, ntID = NULL, angles = c("alpha", "beta",
  "gamma", "delta", "epsilon", "zeta", "chi", "pu_phase"), cex = 0.6,
  cols = 3, file = NULL, width = 15, height = 15, bg = "white",
  units = "cm", res = 200)
```

Arguments

`ntinfo`	A data.frame with the input data. It should contain the columns with the desired angles and a column labeled ntID
`ntID`	A vector of integers with the desired nucleotides of analysis. If NULL all the nucleotides in the data.frame will be used
`angles`	The column names with the desired data
`cex`	To be passed to the `par`.
`cols`	Number of columns in the ouput picture.
`file`	A string with the name of the output file. If NULL, the plot will be printed to screen.

`width`	The width of the plot to be passed to `png`.
`height`	The height of the plot to be passed to `png`.
`bg`	The background color of the plot to be passed to `png`.
`units`	The unit to measure height and width to be passed to `png`.
`res`	Resolution to be passed to `png`.

Value

A series of circular plots with the distributions of the desired angles, which can be directly saved to a ".png" file.

Author(s)

Diego Gallego

Examples

```
ntinfo <- pipeNucData("1bau")
C3endo_ntID <- cleanByPucker(ntinfo, pucker="C3'endo")

## Plot torsional angles for C3'endo nucleotides
plotSetOfDistributions(ntinfo=ntinfo, ntID=C3endo_ntID,
                        file="1bau_C3endo.png")

## Which is the same as doing:
C3endo <- ntinfo[ntinfo$ntID %in% C3endo_ntID,]
plotSetOfDistributions(ntinfo=C3endo, file="1bau_C3endo.png")
```

`queryAPI` *Launch queries to the MMB or EBI APIs*

Description

Given a 4-character string (PDB ID) and the desired "info", it sends a query to the desired API and returns the output. This is an intermediate wrapper called by most of the queryFunctions (for documentation see ?queryFunctions).

Usage

```
queryAPI(ID, info = NULL, API = "default", string1 = NULL,
  string2 = NULL, reuse = TRUE, envir = parent.frame(n = 2),
  verbose = FALSE)
```

Arguments

`ID`	A 4 character string that matches a structure in the Protein Data Bank, or a 3 character string matching a compound.
`info`	A string with the desired query name.
`API`	A string that matches "ebi" or "mmb".
`string1`	A string to configure the query. See example below.

`string2`	A string to configure the query. See example below.
`reuse`	A logical. Set to TRUE if the same query is going to be send repeatedly, so that the result is saved in RAM (ir provides faster user access and avoids unnecessary work in the servers).
`envir`	Environment to save&retrieve the data if reuse is TRUE.
`verbose`	A logical. TRUE to print details of the process.

Value

A vector or data.frame with the desired data.

Author(s)

Diego Gallego

Examples

```
    ## Imagine you want to programmatically access the EBI API contents
    ## through "http://www.ebi.ac.uk/pdbe/api/topology/entry/1s72/chain/H".
    ## 'queryAPI' understands it with four intructions:
    ## 'API="ebi"' stands for the root of the website name ("http.../api/").
    ## 'string1' is the string from the root to the pdb ID.
    ## 'ID' is just the PDB code.
    ## 'string2' is the string after the pdb ID.
    ## Thus, the call would be:
    data <- queryAPI(ID="1s72", API="ebi",
                        string1="topology/entry/", string2="chain/H/")
```

`queryEntryList` *Downloads the list of ID of ALL current PDB entries*

Description

Function to get the list of ALL PDB IDs in the Protein Data Bank at the moment.

Usage

```
queryEntryList()
```

Value

A vector with all the PDB ID entries (updated weekly).

Author(s)

Diego Gallego

Examples

```
pdblist <- queryEntryList()
```

`queryFunctions` *General functions to query PDB (Protein Data Bank) data*

Description

Strightforward way to access structural data by making queries through the EBI or MMB mirrors of the PDB.

Usage

```
queryAuthors(pdbID, ...)

queryChains(pdbID, chain = NULL, subset = NULL, ...)

queryDescription(pdbID, ...)

queryCompType(pdbID, ...)

queryDepdate(pdbID, ...)

queryEntities(pdbID, ...)

queryFormats(pdbID, ...)

queryHeader(pdbID, ...)

queryHetAtms(pdbID, NAtoNa = TRUE, ...)

queryModres(pdbID, onlymodres = FALSE, ...)

queryNDBId(pdbID, ...)

queryLigands(pdbID, onlyligands = FALSE, NAtoNa = TRUE, ...)

queryOrgLigands(pdbID, ...)

queryReldate(pdbID, ...)

queryResol(pdbID, force = FALSE, ...)

queryRevdate(pdbID, ...)

queryStatus(pdbID, ...)

queryTechnique(pdbID, force = FALSE, ...)
```

Arguments

`pdbID`	A 4-character string that matches a structure in the Protein Data Bank.
`...`	For advanced usage, arguments to be passed to subfunction `queryAPI()`.

`chain`	A string with the chain identifier (in case you are only interested in a particular chain). If NULL, the info about all the chains is returned.
`subset`	Optional argument indicating "type", "length" or "description". If NULL, all the columns in the data.frame are returned.
`NAtoNa`	A logical. If TRUE, sodium ion (NA) is modified as "Na".
`onlymodres`	A logical. If TRUE, only the modified residues are returned.
`onlyligands`	A logical. If TRUE, the function only returns the list of unique ligands.
`force`	A logical to force the query to the API (TRUE) or allow checking presaved data.

Value

A character vector or data.frame with the desired information: * queryAuthors List of authors. * queryChains Data frame with list of chains and properties. * queryDescription Author description of the entry. * queryCompType Type of entry as defined in PDB (e.g. Prot-nuc). * queryDepdate Deposition date. * queryEntities Data frame with list of entities and properties. * queryFormats Files available for the entry (e.g. to check if pdb format is available for the structure). * queryHeader Classification of the structure as it appears in the header (PDB format) or in the "_struct_keywords.pdbx_keywords" field (mmCIF format). * queryHetAtms List of HETATM (modified residues and ligands). * queryModres List of modified residues. * queryNDBId NDB ID for Nucleic Acids. * queryLigands Retrieves ligands. * queryOrgLigands Retrieves just the organic ligands (not ions). * queryReldate Release date. * queryResol Resolution * queryRevdate Revision date. * queryStatus PDB status. * queryTechnique Experimental technique

Author(s)

Diego Gallego

Examples

```
queryTechnique("4y4o")
queryAuthors("1s72")
queryNDBId("1bau")
```

`queryObsoleteList` *Downloads the list of ID of ALL current PDB entries*

Description

Function to get the list of Obsolete PDB IDs in the Protein Data Bank at the moment.

Usage

```
queryObsoleteList()
```

Value

A vector with all the PDB ID entries (updated weekly).

Author(s)

Diego Gallego

Examples

```
obsolete <- queryObsoleteList()
```

represAsDataFrame *Coerce Representative list to a data.frame*

Description

Takes the output of getLeontisList or getAltRepres, which represent molecules with the format "XXXX|M|C+XXXX|M|C" (XXXX: PDB ID; M: Model; C: Chain) and returns a data.frame with a more friendly structure:

- Col 1: Equivalence Class.
- Col 2: PDB ID.
- Col 3: Model.
- Col 4: Chain.

Columns 2 to 4 can be the direct input of `pipeNucData()`

Usage

```
represAsDataFrame(nrlist)
```

Arguments

`nrlist`	The output of `getLeontisList()` or `getAltRepres()`.

Value

A data frame with the data of the representative structures

Author(s)

Diego Gallego

Examples

```
data <- getLeontisList(release=3.2, threshold="1.5A")
reps <- represAsDataFrame(nrlist=data)
```

`rVector` *Compute the rVectors between the bases of a RNA structure*

Description

Given a RNA structure it computes the "r" vetors between all bases, as defined by Bottaro et al. 2014 (The role of nucleobase interactions in RNA structure and dynamics). This function is the basis to compute the epsilon RMSD (see the same paper for details and run it with `eRMSD()`).
Furthermore, it also computes two more metrics:
1. The `gamma angle`, as obtained between the x axis of the coordinate system for all the bases projected on the referece base plane and the x axis of the latter. This is a metric of the relative rotation between bases along the orthogonal to the base plane axis (z).
2. The `beta angle`, as obtained between the base planes normal to account for the degree of coplanarity between bases.

Usage

```
rVector(cif, pdb, outformat = "rvector", simple_out = TRUE)

## S4 method for signature 'CIF'
rVector(cif, outformat = "rvector", simple_out = TRUE)

## S4 method for signature 'character'
rVector(cif, outformat = "rvector",
  simple_out = TRUE)

## S4 method for signature 'ANY'
rVector(pdb, outformat = "rvector", simple_out = TRUE)
```

Arguments

`cif`	A CIF object as otained from cifParser.
`pdb`	A pdb object as obtained from cifAsPDB or read.cif/read.pdb (from bio3d package).
`outformat`	A string indicating the output format. This could be: "rvector", "vector_coord" or "cylindrical_coord". "rvector": (r(x)/a, r(y)/a, r(z)/b), being a=5 and b=3. "vector_coord": (r(x), r(y), r(z)). "cylindrical_coord": (rho, phy, z). See reference paper for more details. This does not apply to the `gamma` and `beta` angles.
`simple_out`	A logical to simplify the output to a matrix.

Value

A list of data.frames for the values of each base or a single matrix with all the data appended.

Author(s)

Diego Gallego & Leonardo Darre

Examples

```
cif <- cifParser("2d18")
model1 <- selectModel(cif=cif, model=1)
vectors <- rVector(cif=model1, simple_out=FALSE)
```

---

selectModel *Selects a desired model from a CIF/pdb structure*

Description

Given a object obtained from cifParser/cifAsPDB or read.cif/read.pdb functions (from bio3d specifying "multi = TRUE"), the function returns an object of the same type (CIF or pdb) with the desired model. Since some structures deposited in the PDB contain models with different number of atoms, the pdb objects in R require a special treatment. The cifAsPDB and selectModel functions can cope with these structures (e.g. 1JTW).

Usage

```
selectModel(cif, pdb, model, verbose = FALSE)

## S4 method for signature 'CIF'
selectModel(cif, model, verbose = FALSE)

## S4 method for signature 'ANY'
selectModel(pdb, model, verbose = FALSE)
```

Arguments

`cif`	A CIF object as otained from cifParser.
`pdb`	A pdb object with multiple models (obtained from cifAsPDB or read.cif/read.pdb from bio3d package).
`model`	A string with the desired model number.
`verbose`	A logical to print messages on screen.

Value

A CIF/pdb object with the desired model coordinates.

Author(s)

Diego Gallego

Examples

```
cif <- cifParser("1qfq")
model3 <- selectModel(cif=cif, model=3)
```

`trimByID` *Calls trimSphere to generate smaller pdb files*

Description

Given a data frame with nucleotide info (as obtained from pipeNucData) and the desired nucleotide index (ntID), the function returns a pdb object or file allowing the user to select a number of 5' and 3' neighbors in sequence and non-conected residues in a cutoff radius.

Usage

```
trimByID(cif = NULL, ntID, ntinfo, prev = 2, post = 2,
  verbose = TRUE, file = NULL, ...)
```

Arguments

`cif`	A cif/pdb object obtained from cifParser/read.pdb respectively or a pdb ID so that the function can download the data. If NULL, the function will extract the pdb ID from the ntinfo data frame (pdbID col).
`ntID`	An integer/string with the desired nucleotide ID for analysis.
`ntinfo`	a data.frame with the data. It should contain at least the columns "pdbID", "chain", "model", "resno", "insert" and "ntID" (as the output of pipeNucData function).
`prev`	Number of desired 5' neigbours to be returned.
`post`	Number of desired 3' neigbours to be returned.
`verbose`	A logical to print details of the process.
`file`	A string to save the output in a pdb formated file. If NULL the fucntions just returns the pdb object.
`...`	Arguments to be passed to trimSphere (type ?trimSphere for details)

Value

A smaller pdb object or a pdb file.

Author(s)

Diego Gallego

Examples

```
cif <- cifParser("1bau")
ntinfo <- pipeNucData(cif, torsionals=NULL, distances=NULL, angles=NULL)

## Obtain a smaller pdb of the 4th nucleotide +-2 neigbours and a
## sorrounding sphere of 5 Angstroms
pdb <- trimByID(cif=cif, ntinfo=ntinfo, ntID=4, prev=2, post=2,
                        cutoff=5)

## Same process saving the output in a file:
trimByID(cif=cif, ntinfo=ntinfo, ntID=4, prev=2, post=2,
                        cutoff=5, file="output.pdb")
```

`trimSphere`	*Trim a pdb/cif object to obtain a nucleotide/s of interest and the surrounding area.*

Description

From a pdb/CIF object, the nucleotide of interest and a radius, the function finds all the atoms in the given area and returns a pdb object that only includes the nearest atoms.

Usage

```
trimSphere(cif, model = NULL, ntindex = NULL, chain = NULL,
  sel = NULL, cutoff = 8, cutres = FALSE, file = NULL,
  verbose = TRUE, ...)
```

Arguments

`cif`	A cif/pdb object obtained from cifParser/read.pdb respectively or a pdb ID so that the function can download the data.
`model`	The model of interest to use in the calculations. The first model is always the default.
`ntindex`	A numeric index/indices for the position of the desired nucleotides in the given chain. Not necessary if you provide sel (see below).
`chain`	A string with the chain of interest. Not necessary if you provide sel (see below)..
`sel`	A "select" object as obtained from atom.select (bio3d). Note that if you are using this option, cif must be the same input object you used for the atom.select function.
`cutoff`	A numeric indicating the radius in angstroms to select around the desired nucleotides. If 0 only the nucleotides are returned.
`cutres`	A logical. TRUE to return only what it is found in the cutoff (residues in the boundaries of the cutoff are usually truncated) or FALSE to return whole residues even if further than the cutoff.
`file`	A string to save the output in a pdb formated file. If NULL the fucntions just returns the pdb object.
`verbose`	A logical to print details of the process.
`...`	Arguments to be passed to internal functions.

Value

A smaller pdb object or a pdb file.

Author(s)

Diego Gallego

Examples

```
## Toy example:
cif <- cifParser("1s72")

## Generate a smaller pdb with the residues 55 to 58 of the RNA chain
## "9" with a sorrounding sphere of 5 Angstroms:
smallerpdb <- trimSphere(cif, ntindex=seq(55, 58, 1), chain="9",
                         cutoff=5, verbose=FALSE)

## Same process saving the output in a file:
smallerpdb <- trimSphere(cif, ntindex=seq(55, 58, 1), chain="9",
                         cutoff=5, verbose=FALSE, file="output.pdb")

## Second example:
## Obtain a PDB with just the interacting region between RNA and prot
pdb <- cifAsPDB("1nyb")
data <- findBindingSite(pdb, select="RNA", byres=TRUE)
sel <- bio3d::atom.select(pdb,
                         eleno=append(data$eleno_A, data$eleno_B))
trimSphere(pdb, sel=sel, file="interacting_site.pdb", verbose=FALSE)
```

`veriNA3d` *veriNA3d: Structural bioinformatics package for data mining of the PDB.*

Description

The data pipeline starts obtaining, cleaning and exploring data. When considering biological data, reproducibility, scalability and standardization usually become issues. veriNA3d provides tools to analyse PDB (Protein Data Bank) structural data - with a focus on RNA - while making these issues easy to address. It makes data mining straightforward by providing three categories of functions to get, manage and explore data.

Get/query functions

The query functions ...

Manage/compute functions

The manage functions ...

Explore/save functions

The explore functions ...

Author(s)

Diego Gallego diego.gallego@irbbarcelona.org

Index