Package 'bio3d'

January 27, 2015

Title Biological Structure Analysis

Version 2.1-3

Author Barry Grant, Xin-Qiu Yao, Lars Skjaerven, Julien Ide

VignetteBuilder knitr

Imports parallel, grid

Suggests XML, RCurl, lattice, ncdf, igraph, bigmemory, knitr, testthat (>= 0.9.1)

Depends R (>= 3.1.0)

LazyData yes

Description Utilities to process, organize and explore protein structure, sequence and dynamics data. Features include the ability to read and write structure, sequence and dynamic trajectory data, perform sequence and structure database searches, data summaries, atom selection, alignment, superposition, rigid core identification, clustering, torsion analysis, distance matrix analysis, structure and sequence conservation analysis, normal mode analysis, principal component analysis of heterogeneous structure data, and correlation network analysis from normal mode and molecular dynamics data. In addition, various utility functions are provided to enable the statistical and graphical power of the R environment to work with biological sequence and structural data. Please refer to the URLs below for more information.

Maintainer Barry Grant <bjgrant@umich.edu>

License GPL (>= 2)

URL http://thegrantlab.org/bio3d/, http://bitbucket.org/Grantlab/bio3d

NeedsCompilation no

Repository CRAN

Date/Publication 2014-10-27 19:03:11

R topics documented:

`bio3d-package` *Biological Structure Analysis*

Description

Utilities for the analysis of protein structure and sequence data.

Details

Package:	bio3d
Type:	Package
Version:	2.1-3
Date:	2014-10-24
License:	GPL version 2 or newer
URL:	http://thegrantlab.org/bio3d/

Features include the ability to read and write structure (`read.pdb`, `write.pdb`, `read.fasta.pdb`), sequence (`read.fasta`, `write.fasta`) and dynamics trajectory data (`read.dcd`, `read.ncdf`, `write.ncdf`).

Perform sequence and structure database searches (`blast.pdb`, `hmmer`), atom summaries (`summary.pdb`), atom selection (`atom.select`), alignment (`pdbaln`, `seqaln`, `mustang`) superposition (`rot.lsq`, `fit.xyz`), rigid core identification (`core.find`, `plot.core`, `fit.xyz`), dynamic domain analysis (`geostas`), torsion/dihedral analysis (`torsion.pdb`, `torsion.xyz`), clustering (via `hclust`), principal component analysis (`pca.xyz`, `pca.pdbs`, `pca.tor`, `plot.pca`, `plot.pca.loadings`, `mktrj.pca`), dynamical cross-correlation analysis (`dccm`, `lmi`, `plot.dccm`) and correlation network analysis (`cna`, `plot.cna`) of structure data.

Perform conservation analysis of sequence (`seqaln`, `conserv`, `seqidentity`, `entropy`, `consensus`) and structural (`pdbaln`, `rmsd`, `rmsf`, `core.find`) data.

Perform normal mode analysis (nma, build.hessian), ensemble normal mode analysis (nma.pdbs), mode comparison (rmsip) and (overlap), atomic fluctuation prediction (fluct.nma), cross-correlation analysis (dccm.nma), cross-correlation visualization (view.dccm), deformation analysis (deformation.nma), and mode visualization (view.modes), (mktrj.nma).

In addition, various utility functions are provided to facilitate manipulation and analysis of biological sequence and structural data (e.g. get.pdb, get.seq, aa123, aa321, pdbseq, aln2html, atom.select, rot.lsq, fit.xyz, is.gap, gap.inspect, orient.pdb, pairwise, plot.bio3d, plot.nma, plot.blast, etc.).

Note

The latest version, package vignettes and documentation with worked example outputs can be obtained from the bio3d website:
http://thegrantlab.org/bio3d/.
http://thegrantlab.org/bio3d/html/.
http://bitbucket.org/Grantlab/bio3d.

Author(s)

Barry Grant <bjgrant@umich.edu> Xin-Qiu Yao <xinqyao@umich.edu> Lars Skjaerven <larsss@gmail.com> Julien Ide <julien.ide.fr@gmail.com>

References

Grant, B.J. et al. (2006) *Bioinformatics* **22**, 2695–2696.

Examples

```
help(package="bio3d")     # list the functions within the package
#lbio3d()                  # list bio3d function names only

## Or visit:
##   http://thegrantlab.org/bio3d/html/

## See the individual functions for further documentation and examples, e.g.
#help(read.pdb)

## Or online:
##    http://thegrantlab.org/bio3d/html/read.pdb.html

## Not run:
##-- See the list of Bio3D demos
demo(package="bio3d")

## Try some out, e.g:
demo(pdb) # PDB Reading, Manipulation, Searching and Alignment
demo(pca) # Principal Component Analysis
demo(md)  # Molecular Dynamics Trajectory Analysis
demo(nma) # Normal Mode Analysis
```

```
##-- Start by reading a PDB file
pdb <- read.pdb("1BG2")

##-- Print summary information
pdb

##
##-- Distance matrix
##
k <- dm(pdb, selection="calpha")
plot(k)

## Extract SEQRES PDB sequence
s1 <- aa321(pdb$seqres)

## Extract ATOM PDB sequence
s2 <- pdbseq(pdb)

## write a FASTA format sequence file
write.fasta(seqs=seqbind(s1, s2), id=c("seqres","atom"), file="eg.fa")

##----------------------------------------##

##
##-- Select alpha carbon atom subset
##
ca.inds <- atom.select(pdb, "calpha")

## Plot of B-factor values along with secondary structure
plot.bio3d(pdb$atom[ca.inds$atom, "b"], sse=pdb, ylab="B-factor")

## Secondary structure assignment with DSSP
#sse <- dssp(pdb)

##----------------------------------------##

##
##-- Torsion angle analysis and basic Ramachandran plot
##
tor <- torsion.pdb(pdb)
plot(tor$phi, tor$psi)

##----------------------------------------##

##
##-- Search for related structures in the PDB database
blast <- blast.pdb( pdbseq(pdb) )
hits  <- plot.blast(blast)
head(hits$hits)

## Download these with function "get.pdb()"
#rawpdbs <- get.pdb( hits$pdb.id, "PDB_downloads")
```

```
## Split by chain with function "pdbsplit()"
#pdbsplit(rawpdbs, path="PDB_downloads/split_chain")

## and then align with "pdbaln()" and superpose with "pdbfit()"
#hitfiles <- paste0("PDB_downloads/split_chain", hits$pdb.id, ".pdb")
#pdbs <- pdbaln(hitfiles)
#xyz  <- pdbfit(pdbs)

##--------------------------------------##

##
##-- Read an example FASTA sequence alignment from PFAM
##
infile <- "http://pfam.sanger.ac.uk/family/PF00071/alignment/seed/format?format=fasta"

aln <- read.fasta( infile )

## Entropy and similarity scores for alignment positions
h <- entropy(aln)
s <- conserv(aln)   # see other"conserv()" options
plot(h$H.norm, typ="h", ylab="Normalized entropy score", col="gray")
points( s, typ="h", col="red")

## Alignment consensus sequence
con <- consensus(aln)
con$seq

## add consensus sequence to conservation plot
ind <- which(s > 0.6)
text(ind, s[ind], labels=con$seq[ind])

## Render the alignment as coloured HTML
aln2html(aln, append=FALSE, file="eg.html")

##--------------------------------------##

##
##-- Read an alignment of sequences and their corresponding structures
##
aln  <- read.fasta( system.file("examples/kif1a.fa", package="bio3d") )
pdbs <- read.fasta.pdb( aln )

##-- DDM: Difference Distance Matrix
a <- dm(pdbs$xyz[2,])
b <- dm(pdbs$xyz[3,])
ddm <- a - b
plot(ddm,key=FALSE, grid=FALSE)


##-- Superpose structures on non gap positions
xyz <- pdbfit(pdbs)

##-- RMSD of non gap positions
```

```
gaps <- gap.inspect(pdbs$xyz)
rmsd(pdbs$xyz[, gaps$f.inds])
rmsd(xyz[, gaps$f.inds])


##-- Rigid 'core' identification
core <- core.find(pdbs)
#plot(core)

## Core fit the structures (superpose on rigid zones)
xyz2 <- pdbfit(pdbs, inds=core$c0.5A.xyz)

## Note larger overall RMSD but lower core-residue RMSF
rmsd(xyz2[, gaps$f.inds])

plot(rmsf(xyz), typ="l", col="blue", ylab="RMSF")
points(rmsf(xyz2), typ="l", col="red")



##
##-- PCA of experimental structures
##
# Ignore gap containing positions
gaps.res <- gap.inspect(pdbs$ali)
gaps.pos <- gap.inspect(pdbs$xyz)


##-- Do PCA
pc.xray <- pca.xyz(xyz[, gaps.pos$f.inds])
## Or more simply
# pc.xray <- pca.xyz(xyz, rm.gaps=TRUE)

## Plot results
plot(pc.xray)

plot.pca.loadings(pc.xray$au)

## Write a PC trajectory (for viewing as tube in VMD)
rn <- pdbs$resno[1, gaps.res$f.inds]
rd <- aa123(pdbs$ali[1, gaps.res$f.inds])

#p1 <- mktrj.pca(pc.xray, pc=1, resno =rn, resid = rd, file="pc1.pdb")
#p2 <- mktrj.pca(pc.xray, pc=2, resno =rn, resid = rd, file="pc2.pdb")
#p3 <- mktrj.pca(pc.xray, pc=3, resno =rn, resid = rd, file="pc3.pdb")


##----------------------------------------##

##
##-- Read a CHARMM/X-PLOR/NAMD trajectory file
##
trtfile <- system.file("examples/hivp.dcd", package="bio3d")
```

```
trj <- read.dcd(trtfile)

## Read the starting PDB file to determine atom correspondence
pdbfile <- system.file("examples/hivp.pdb", package="bio3d")
pdb <- read.pdb(pdbfile)

## Fit trj on PDB based on residues 23 to 31 and 84 to 87 in both chains
inds <- atom.select(pdb, resno=c(23:31,84:87), elety="CA")
fit.xyz <- fit.xyz(pdb$xyz, trj, fixed.inds=inds$xyz, mobile.inds=inds$xyz)

##-- RMSD of trj frames from PDB
r <- rmsd(a=pdb, b=fit.xyz)

##-- PCA of trj
pc.trj <- pca.xyz(fit.xyz)

## Plot PCA results
plot(pc.trj)

## Examine residue-wise contributions to PCs
plot.pca.loadings(pc.trj$au)

## cluster in PC1 subspace
hc <- hclust(dist(pc.trj$z[,1]))
plot(pc.trj, col=cutree(hc, k=2))


## Write PC trajectory for viewing as tube in VMD
a <- mktrj.pca(pc.trj, pc=1, file="pc1_trj.pdb")


## End(Not run)
## Other examples include: normal mode analysis, alignment, clustering etc...
## See: http://thegrantlab.org/bio3d/tutorials
```

`aa.index` *AAindex: Amino Acid Index Database*

Description

A collection of published indices, or scales, of numerous physicochemical and biological properties of the 20 standard aminoacids (Release 9.1, August 2006).

Usage

```
data(aa.index)
```

Format

A list of 544 named indeces each with the following components:

1. H character vector: Accession number.
2. D character vector: Data description.
3. R character vector: LITDB entry number.
4. A character vector: Author(s).
5. T character vector: Title of the article.
6. J character vector: Journal reference.
7. C named numeric vector: Correlation coefficients of similar indeces (with coefficients of 0.8/-0.8 or more/less). The correlation coefficient is calculated with zeros filled for missing values.
8. I named numeric vector: Amino acid index data.

Source

'AAIndex' was obtained from:
ftp://ftp.genome.ad.jp/pub/db/genomenet/aaindex/aaindex1
For a description of the 'AAindex' database see:
http://www.genome.jp/aaindex/.

References

Grant, B.J. et al. (2006) *Bioinformatics* **22**, 2695–2696.

'AAIndex' is the work of Kanehisa and co-workers:
Kawashima and Kanehisa (2000) *Nucleic Acids Res.* **28**, 374;
Tomii and Kanehisa (1996) *Protein Eng.* **9**, 27–36;
Nakai, Kidera and Kanehisa (1988) *Protein Eng.* **2**, 93–100.

Examples

```
## Load AAindex data
data(aa.index)

## Find all indeces described as "volume"
ind <- which(sapply(aa.index, function(x)
                    length(grep("volume", x$D, ignore.case=TRUE)) != 0))

## find all indeces with author "Kyte"
ind <- which(sapply(aa.index, function(x) length(grep("Kyte", x$A)) != 0))

## examine the index
aa.index[[ind]]$I

## find indeces which correlate with it
all.ind <- names(which(Mod(aa.index[[ind]]$C) >= 0.88))

## examine them all
sapply(all.ind, function (x) aa.index[[x]]$I)
```

aa.mass *Amino Acid Residue Mass*

Description

This data set provides the atomic masses of a selection of amino acids regularly occuring in proteins.

Usage

```
aa.mass
```

Format

A data frame with the following components.

`aa3` a character vector containing three-letter amino acid code.

`aa1` a character vector containing one-letter amino acid code.

`aaMass` a numeric vector containing the mass of the respective amino acids.

`formula` a character vector containing the formula of the amino acid in which the mass calculat was based.

`name` a character vector containing the full names of the respective amino acids.

References

Grant, B.J. et al. (2006) *Bioinformatics* **22**, 2695–2696.

See Also

aa2mass, aa.index, atom.index, elements,

Examples

```
data(aa.mass)
aa.mass

## table look up
aa.mass["HIS", ]

## read PDB, and fetch residue masses
pdb <- read.pdb("1etl")
aa2mass(pdb)
```

aa123 *Convert Between 1-letter and 3-letter Aminoacid Codes*

Description

Convert between one-letter IUPAC aminoacid codes and three-letter PDB style aminoacid codes.

Usage

```
aa123(aa)
aa321(aa)
```

Arguments

aa	a character vector of individual aminoacid codes.

Details

Standard conversions will map 'A' to 'ALA', 'G' to 'GLY', etc. Non-standard codes in aa will generate a warning and return 'UNK' or 'X'.

Value

A character vector of aminoacid codes.

Author(s)

Barry Grant

References

Grant, B.J. et al. (2006) *Bioinformatics* **22**, 2695–2696.

For a description of IUPAC one-letter codes see:
http://www.chem.qmul.ac.uk/iupac/AminoAcid/

For a description of PDB residue codes see Appendix 4:
http://msdlocal.ebi.ac.uk/docs/pdb_format/appendix.html

See Also

read.pdb, read.fasta

Examples

```
# Simple conversion
aa123(c("D","L","A","G","S","H"))
aa321(c("ASP", "LEU", "ALA", "GLY", "SER", "HIS"))

## Not run:
# Extract sequence from PDB file's ATOM and SEQRES cards
pdb <- read.pdb( "http://www.rcsb.org/pdb/files/1BG2.pdb" )
s <- aa321(pdb$seqres)                    # SEQRES
a <- aa321(pdb$atom[pdb$calpha,"resid"]) # ATOM

# Write both sequences to fasta file
write.fasta(id=c("seqres","atom"), seqs=seqbind(s,a), file="eg2.fa")

## End(Not run)
```

`aa2index` *Convert an Aminoacid Sequence to AAIndex Values*

Description

Converts sequences to aminoacid indeces from the 'AAindex' database.

Usage

```
aa2index(aa, index = "KYTJ820101", window = 1)
```

Arguments

`aa`	a protein sequence character vector.
`index`	an index name or number (default: "KYTJ820101", hydropathy index by Kyte-Doolittle, 1982).
`window`	a positive numeric value, indicating the window size for smoothing with a sliding window average (default: 1, i.e. no smoothing).

Details

By default, this function simply returns the index values for each amino acid in the sequence. It can also be set to perform a crude sliding window average through the `window` argument.

Value

Returns a numeric vector.

Author(s)

Ana Rodrigues

References

Grant, B.J. et al. (2006) *Bioinformatics* **22**, 2695–2696.

'AAIndex' is the work of Kanehisa and co-workers: Kawashima and Kanehisa (2000) *Nucleic Acids Res.* **28**, 374; Tomii and Kanehisa (1996) *Protein Eng.* **9**, 27–36; Nakai, Kidera and Kanehisa (1988) *Protein Eng.* **2**, 93–100.

For a description of the 'AAindex' database see:
http://www.genome.jp/aaindex/ or the aa.index documentation.

See Also

aa.index, read.fasta

Examples

```
## Residue hydropathy values
seq <- c("R","S","D","X","-","X","R","H","Q","V","L")
aa2index(seq)

## Not run:
## Use a sliding window average
aa2index(aa=seq, index=22, window=3)

## Use an alignment

aln  <- read.fasta(system.file("examples/hivp_xray.fa",package="bio3d"))
prop <- t(apply(aln$ali, 1, aa2index, window=1))

## find and use indices for volume calculations
i <- which(sapply(aa.index,
       function(x) length(grep("volume", x$D, ignore.case=TRUE)) != 0))
sapply(i, function(x) aa2index(aa=seq, index=x, window=5))

## End(Not run)
```

---

aa2mass *Amino Acid Residues to Mass Converter*

Description

Convert a sequence of amino acid residue names to mass.

Usage

```
aa2mass(pdb, inds=NULL, mass.custom=NULL, addter=TRUE, mmtk=FALSE)
```

Arguments

pdb	a character vector containing the atom names to convert to atomic masses. Alternatively, a object of type pdb can be provided.
inds	atom and xyz coordinate indices obtained from atom.select that selects the elements of pdb upon which the calculation should be based.
mass.custom	a list of amino acid residue names and their corresponding masses.
addter	logical, if TRUE terminal atoms are added to final masses.
mmtk	logical, if TRUE use the exact aminoacid residue masses as provided with the MMTK database (for testing purposes).

Details

This function converts amino acid residue names to their corresponding masses. In the case of a non-standard amino acid residue name mass.custom can be used to map the residue to the correct mass. User-defined amino acid masses (with argument mass.custom) will override mass entries obtained from the database.

See examples for more details.

Value

Returns a numeric vector of masses.

Note

When object of type pdb is provided, non-calpha atom records are omitted from the selection.

Author(s)

Lars Skjaerven

References

Grant, B.J. et al. (2006) *Bioinformatics* **22**, 2695–2696.

See Also

atom.index, atom2mass, aa.index

Examples

```
resi.names <- c("LYS", "ALA", "CYS", "HIS")
masses <- aa2mass(resi.names, addter=FALSE)

## Not run:
## Fetch atomic masses in a PDB object
pdb <- read.pdb("3dnd")
masses <- aa2mass(pdb)

## or
```

```
masses <- aa2mass(pdb$atom[1:10,"resid"])

## Dealing with unconventional residues
#pdb <- read.pdb("1xj0")

#mass.cust <- list("CSX"=122.166)
#masses <- aa2mass(pdb, mass.custom=mass.cust)

## End(Not run)
```

`aln2html` *Create a HTML Page For a Given Alignment*

Description

Renders a sequence alignment as coloured HTML suitable for viewing with a web browser.

Usage

```
aln2html(aln, file="alignment.html", Entropy=0.5, append=TRUE,
         caption.css="color: gray; font-size: 9pt",
         caption="Produced by <a href=http://thegrantlab.org/bio3d/>Bio3D</a>",
         fontsize="11pt", bgcolor=TRUE, colorscheme="clustal")
```

Arguments

`aln`	an alignment list object with `id` and `ali` components, similar to that generated by `read.fasta`.
`file`	name of output html file.
`Entropy`	conservation 'cuttoff' value below which alignment columns are not coloured.
`append`	logical, if TRUE output will be appended to `file`; otherwise, it will overwrite the contents of `file`.
`caption.css`	a character string of css options for rendering 'caption' text.
`caption`	a character string of text to act as a caption.
`fontsize`	the font size for alignment characters.
`bgcolor`	background colour.
`colorscheme`	conservation colouring scheme, currently only "clustal" is supported with alternative arguments resulting in an entropy shaded alignment.

Value

Called for its effect.

Note

Your web browser should support style sheets.

Author(s)

Barry Grant

References

Grant, B.J. et al. (2006) *Bioinformatics* **22**, 2695–2696.

See Also

read.fasta, write.fasta, seqaln

Examples

```
## Not run:
## Read an example alignment
aln <- read.fasta(system.file("examples/hivp_xray.fa",package="bio3d"))

## Produce a HTML file for this alignment
aln2html(aln, append=FALSE, file=file.path("eg.html")
aln2html(aln, colorscheme="ent", file="eg.html")
## View/open the file in your web browser
#browseURL("eg.html")

## End(Not run)
```

---

angle.xyz *Calculate the Angle Between Three Atoms*

Description

A function for basic bond angle determination.

Usage

```
angle.xyz(xyz, atm.inc = 3)
```

Arguments

xyz	a numeric vector of Cartisean coordinates.
atm.inc	a numeric value indicating the number of atoms to increment by between successive angle evaluations (see below).

Value

Returns a numeric vector of angles.

Note

With `atm.inc=1`, angles are calculated for each set of three successive atoms contained in `xyz` (i.e. moving along one atom, or three elements of `xyz`, between sucessive evaluations). With `atm.inc=3`, angles are calculated for each set of three successive non-overlapping atoms contained in `xyz` (i.e. moving along three atoms, or nine elements of `xyz`, between sucessive evaluations).

Author(s)

Barry Grant

References

Grant, B.J. et al. (2006) *Bioinformatics* **22**, 2695–2696.

See Also

`torsion.pdb`, `torsion.xyz`, `read.pdb`, `read.dcd`.

Examples

```
## Read a PDB file
pdb <- read.pdb( "http://www.rcsb.org/pdb/files/1BG2.pdb" )

## Angle between N-CA-C atoms of residue four
inds <- atom.select(pdb,"///4///N,CA,C/")
angle.xyz(pdb$xyz[inds$xyz])

## Basic stats of all N-CA-C bound angles
inds <- atom.select(pdb,"//////N,CA,C/")
summary( angle.xyz(pdb$xyz[inds$xyz]) )
#hist( angle.xyz(pdb$xyz[inds$xyz]), xlab="Angle" )
```

`atom.index` *Atom Names/Types*

Description

This data set gives for various atom names/types the corresponding atomic symbols.

Usage

```
atom.index
```

Format

A data frame with the following components.

`name` a character vector containing atom names/types.

`symb` a character vector containing atomic symbols.

References

Grant, B.J. et al. (2006) *Bioinformatics* **22**, 2695–2696.

See Also

`elements`, `atom.index`, `atom2ele`

Examples

```
data(atom.index)
atom.index

# Get the atomic symbol of some atoms
atom.names <- c("CA", "O", "N", "OXT")
atom.index[match(atom.names, atom.index$name), "symb"]
```

---

atom.select *Atom Selection From PDB Structure*

Description

Return the 'atom' and 'xyz' coordinate indices of a 'pdb' structure object corresponding to the intersection of a hierarchical selection.

Usage

```
atom.select(pdb, string=NULL, chain=NULL, resno=NULL, resid=NULL,
eleno=NULL, elety=NULL, type=NULL, verbose=TRUE, rm.insert=FALSE)
## S3 method for class 'select'
print(x, ...)
```

Arguments

pdb	a structure object of class "pdb", obtained from `read.pdb`.
string	a character selection string with the following syntax: `/segid/chain/resno/resid/eleno/elety/`. Or a single selection keyword from `calpha cbeta backbone protein notprotein ligand water notwater h noh`

chain	a character vector of chain identifiers.
resno	a numeric or character vector of residue numbers.
resid	a character vector of residue name identifiers.
eleno	a numeric or character vector of element numbers.
elety	a character vector of atom names.
type	a single element character vector for selecting 'ATOM' or 'HETATM' record types.
verbose	logical, if TRUE details of the selection are printed.
rm.insert	logical, if TRUE insert ATOM records from the pdb object are ignored.
x	a atom.select object as obtained from atom.select.
...	additional arguments to 'print'. Currently ignored.

Details

This function allows for the selection of atom and coordinate data corresponding to the intersection of various input criteria.

Input selection criteria include selection `string` keywords (such as `"calpha"`, `"backbone"`, `"protein"`, `"ligand"`, etc.) and individual named selection components (including 'chain', 'resno', 'resid', 'elety' etc.). For example, `atom.select(pdb, "calpha")` or `atom.select(pdb, elety="CA")`, which in this case are equivalent.

Alternatively, a single element character vector containing a hierarchical 'selection string', `string`, with a strict format composed of six sections separated by a '/' character can be used.

Each section of this 'selection string' corresponds to a different level in the PDB structure hierarchy, namely: (1) segment identifier, (2) chain identifier, (3) residue number, (4) residue name, (5) element number, and (6) element name.

For example, the string `//A/65:143///CA/` selects all C-alpha atoms from residue numbers 65 to 143, of chain A.

A simpler alternative would be `atom.select(pdb, chain="A", resno=65:143, elety="CA")`.

More typical use is through keyword `string` shortcuts `"calpha"`, `"back"`, `"backbone"`, `"cbeta"`, `"protein"`, `"notprotein"`, `"ligand"`, `"water"`, `"notwater"`, `"h"` and `"noh"`.

Note that keyword `string` shortcuts can be combined with individual selection components, e.g. `atom.select(pdb, "protein", chain="A")`. See below for further examples.

In addition, the `combine.sel` function can further combine atom selections using 'and', 'or', or 'not' logical operations.

When called without selection criteria, `atom.select` will print a summary of pdb makeup.

Value

Returns a list of class `"select"` with components:

atom	atom indices of selected atoms.
xyz	xyz indices of selected atoms.

Author(s)

Barry Grant

References

Grant, B.J. et al. (2006) *Bioinformatics* **22**, 2695–2696.

See Also

combine.sel, read.pdb, write.pdb, read.dcd

Examples

```
# Read a PDB file
pdb <- read.pdb( system.file("examples/1hel.pdb", package="bio3d") )

# Print structure summary
atom.select(pdb)

# Select all C-alpha atoms with residues numbers between 43 and 54
ca.inds <- atom.select(pdb, "calpha", resno=43:54)
print( pdb$atom[ ca.inds$atom, "resid" ] )
print( pdb$xyz[ ca.inds$xyz ] )

## Not run:
# Select all C-alphas
ca.inds <- atom.select(pdb, "calpha")

# String examples (see above).
ca.inds <- atom.select(pdb, "///43:54///CA/")
inds <- atom.select(pdb, "///107:118///N,CA,C,O/")

## End(Not run)
```

---

atom2ele *Atom Names/Types to Atomic Symbols Converter*

Description

Convert atom names/types into atomic symbols

Usage

```
atom2ele(...)

## Default S3 method:
atom2ele(x, elety.custom=NULL, rescue=TRUE, ...)
```

```
## S3 method for class 'pdb'
atom2ele(pdb, inds, elety.custom=NULL, rescue=TRUE, ...)
```

Arguments

x	a character vector containing atom names/types to be converted.
elety.custom	a customized data.frame containing atom names/types and corresponding atomic symbols.
rescue	logical, if TRUE the atomic symbols will be mapped to the first character of the atom names/types.
pdb	an object of class 'pdb' for which elety will be converted.
inds	an object of class 'select' indicating a subset of the pdb object to be used (see atom.select and trim.pdb).
...	further arguments passed to or from other methods.

Details

The default method searchs for the atom names/types in the atom.index data set and returns their corresponding atomic symbols. If elety.custom is specified it is combined with atom.index (using rbind) before searching. Therefore, elety.custom must contains columns named name and symb.

The S3 method for object of class 'pdb', pass pdb$atom[,"elety"] to the default method.

Value

Return a character vector of atomic symbols

Author(s)

Julien Ide, Lars Skjaerven

See Also

atom.index, elements, read.pdb, atom2mass, formula2mass

Examples

```
atom.names <- c("CA", "O", "N", "OXT")
atom2ele(atom.names)


## Get atomic symbols from a PDB object with a customized data set
pdb <- read.pdb("3RE0",verbose=FALSE)
inds <- atom.select(pdb, resno=201, verbose=FALSE)

## maps CL2 to C
atom2ele(pdb, inds, elety.custom = NULL)

## map CL2 to Cl manually
```

```
myelety <- data.frame(name = "CL2", symb = "Cl")
atom2ele(pdb, inds, elety.custom = myelety)
```

---

atom2mass *Atom Names/Types to Mass Converter*

Description

Convert atom names/types into atomic masses.

Usage

```
atom2mass(...)
## Default S3 method:
atom2mass(x, mass.custom=NULL, elety.custom=NULL,
                            grpby=NULL, rescue=TRUE, ...)
## S3 method for class 'pdb'
atom2mass(pdb, inds=NULL, mass.custom=NULL,
                        elety.custom=NULL, grpby=NULL, rescue=TRUE, ...)
```

Arguments

x	a character vector containing atom names/types to be converted.
mass.custom	a customized data.frame containing atomic symbols and corresponding masses.
elety.custom	a customized data.frame containing atom names/types and corresponding atomic symbols.
grpby	a 'factor', as returned by as.factor, used to group the atoms.
rescue	logical, if TRUE the atomic symbols will be mapped to the first character of the atom names/types.
pdb	an object of class 'pdb' for which elety will be converted.
inds	an object of class 'select' indicating a subset of the pdb object to be used (see atom.select and trim.pdb).
...	.

Details

The default method first convert atom names/types into atomic symbols using the atom2ele function. Then, atomic symbols are searched in the elements data set and their corresponding masses are returned. If mass.custom is specified it is combined with elements (using rbind) before searching. Therefore, mass.custom must have columns named symb and mass (see examples). If grpby is specified masses are splitted (using split) to compute the mass of groups of atoms defined by grpby.

The S3 method for object of class 'pdb', pass pdb$atom$elety to the default method.

Value

Return a numeric vector of masses.

Author(s)

Julien Ide, Lars Skjaerven

See Also

elements, atom.index, atom2ele, read.pdb

Examples

```
atom.names <- c("CA", "O", "N", "OXT")
atom2mass(atom.names)



## Get atomic symbols from a PDB object with a customized data set
pdb <- read.pdb("3RE0", verbose=FALSE)
inds <- atom.select(pdb, resno=201, verbose=FALSE)

## selected atoms
print(pdb$atom$elety[inds$atom])

## default will map CL2 to C
atom2mass(pdb, inds)

## map element CL2 correctly to Cl
myelety  <- data.frame(name = c("CL2","PT1","N1","N2"), symb = c("Cl","Pt","N","N"))
atom2mass(pdb, inds, elety.custom = myelety)

## custom masses
mymasses <- data.frame(symb = c("Cl","Pt"), mass = c(35.45, 195.08))
atom2mass(pdb, inds, elety.custom = myelety, mass.custom = mymasses)
```

---

atom2xyz *Convert Between Atom and xyz Indices*

Description

Basic functions to convert between xyz and their corresponding atom indices.

Usage

```
atom2xyz(num)
xyz2atom(xyz.ind)
```

Arguments

num	a numeric vector of atom indices.
xyz.ind	a numeric vector of xyz indices.

Value

A numeric vector of either xyz or atom indices.

Author(s)

Barry Grant

References

Grant, B.J. et al. (2006) *Bioinformatics* **22**, 2695–2696.

See Also

atom.select, read.pdb

Examples

```
xyz.ind <- atom2xyz(c(1,10,15))
xyz2atom( xyz.ind )
```

---

bhattacharyya *Bhattacharyya Coefficient*

Description

Calculate the Bhattacharyya Coefficient as a similarity between two modes objects.

Usage

```
bhattacharyya(...)
## S3 method for class 'enma'
bhattacharyya(enma, covs=NULL, ncore=NULL, ...)
## S3 method for class 'array'
bhattacharyya(covs, ncore=NULL, ...)
## S3 method for class 'matrix'
bhattacharyya(a, b, q=90, n=NULL, ...)
## S3 method for class 'nma'
bhattacharyya(...)
## S3 method for class 'pca'
bhattacharyya(...)
```

Arguments

enma	an object of class "enma" obtained from function nma.pdbs.
covs	an array of covariance matrices of equal dimensions.
ncore	number of CPU cores used to do the calculation. ncore>1 requires package 'parallel' installed.
a	covariance matrix to be compared with b.
b	covariance matrix to be compared with a.
q	a numeric value (in percent) determining the number of modes to be compared.
n	the number of modes to be compared.
...	arguments passed to associated functions.

Details

Bhattacharyya coefficient provides a means to compare two covariance matrices derived from NMA or an ensemble of conformers (e.g. simulation or X-ray conformers).

Value

Returns the similarity coefficient(s).

Author(s)

Lars Skjaerven

References

Grant, B.J. et al. (2006) *Bioinformatics* **22**, 2695–2696. Fuglebakk, E. et al. (2013) *JCTC* **9**, 5618–5628.

See Also

Other similarity measures: sip, covsoverlap, rmsip.

---

binding.site *Binding Site Residues*

Description

Determines the interacting residues between two PDB entities.

Usage

```
binding.site(a, b=NULL, a.inds=NULL, b.inds=NULL, cut=5, hydrogens=TRUE)
```

Arguments

a	an object of class pdb as obtained from function read.pdb.
b	an object of class pdb as obtained from function read.pdb.
a.inds	atom and xyz coordinate indices obtained from atom.select that selects the elements of a upon which the calculation should be based.
b.inds	atom and xyz coordinate indices obtained from atom.select that selects the elements of b upon which the calculation should be based.
cut	distance cutoff
hydrogens	logical, if FALSE hydrogen atoms are omitted from the calculation.

Details

This function reports the residues of a closer than a cutoff to b. This is a wrapper function calling the underlying function dm.xyz.

If b=NULL then b.inds should be elements of a upon which the calculation is based (typically chain A and B of the same PDB file).

Value

Returns a list with the following components:

atom.inds	atom indices of a.
xyz.inds	xyz indices of a.
resnames	a character vector of interacting residues.
resno	a numeric vector of interacting residues numbers.

Author(s)

Lars Skjaerven

References

Grant, B.J. et al. (2006) *Bioinformatics* **22**, 2695–2696.

See Also

read.pdb, atom.select, dm

Examples

```
pdb <- read.pdb('3dnd')

# Binding site residues
rec.inds <- atom.select(pdb, string='//A/1:350////')
lig.inds <- atom.select(pdb, string='//A/351////')
bs <- binding.site(pdb, a.inds=rec.inds, b.inds=lig.inds)
```

```
     # Interaction between peptide and protein
     rec.inds <- atom.select(pdb, string='//A/1:350////')
     lig.inds <- atom.select(pdb, string='//I/5:24////')
     bs <- binding.site(pdb, a.inds=rec.inds, b.inds=lig.inds)


     ## Not run:
     # Interaction between two PDB entities
#     rec <- read.pdb("receptor.pdb")
#     lig <- read.pdb("ligand.pdb")
     rec <- trim.pdb(pdb, inds=rec.inds)
     lig <- trim.pdb(pdb, inds=lig.inds)
     bs <- binding.site(rec, lig, hydrogens=FALSE)

## End(Not run)
```

`blast.pdb` *NCBI BLAST Sequence Search*

Description

Run NCBI blastp, on a given sequence, against the PDB, NR and swissprot sequence databases.

Usage

```
blast.pdb(seq, database = "pdb", time.out = NULL, chain.single=TRUE)
get.blast(urlget, time.out = NULL, chain.single=TRUE)
```

Arguments

`seq`	a single element or multi-element character vector containing the query sequence. Alternatively a 'fasta' object from function `get.seq` can be provided.
`database`	a single element character vector specifying the database against which to search. Current options are 'pdb', 'nr' and 'swissprot'.
`time.out`	integer specifying the number of seconds to wait for the blast reply before a time out occurs.
`urlget`	the URL to retrieve BLAST results; Usually it is returned by blast.pdb if time.out is set and met.
`chain.single`	logical, if TRUE double NCBI character PDB database chain identifiers are simplified to lowercase '1WF4_GG' > '1WF4_g'. If FALSE no conversion to match RCSB PDB files is performed.

Details

This function employs direct HTTP-encoded requests to the NCBI web server to run BLASTP, the protein search algorithm of the BLAST software package.

BLAST, currently the fastest and most popular pairwise sequence comparison algorithm, performs gapped local alignments, through the implementation of a heuristic strategy: it identifies short nearly exact matches or hits, bidirectionally extends non-overlapping hits resulting in ungapped extended hits or high-scoring segment pairs (HSPs), and finally extends the highest scoring HSP in both directions via a gapped alignment (Altschul et al., 1997)

For each pairwise alignment BLAST reports the raw score, bitscore and an E-value that assess the statistical significance of the raw score. Note that unlike the raw score E-values are normalized with respect to both the substitution matrix and the query and database lengths.

Here we also return a corrected normalized score (mlog.evalue) that in our experience is easier to handle and store than conventional E-values. In practice, this score is equivalent to minus the natural log of the E-value. Note that, unlike the raw score, this score is independent of the substitution matrix and and the query and database lengths, and thus is comparable between BLASTP searches.

Value

A list with eight components:

`bitscore`	a numeric vector containing the raw score for each alignment.
`evalue`	a numeric vector containing the E-value of the raw score for each alignment.
`mlog.evalue`	a numeric vector containing minus the natural log of the E-value.
`gi.id`	a character vector containing the gi database identifier of each hit.
`pdb.id`	a character vector containing the PDB database identifier of each hit.
`hit.tbl`	a character matrix summarizing BLAST results for each reported hit, see below.
`raw`	a data frame summarizing BLAST results, note multiple hits may appear in the same row.
`url`	a single element character vector with the NCBI result URL and RID code. This can be passed to the get.blast function.

Note

Online access is required to query NCBI blast services.

Author(s)

Barry Grant

References

Grant, B.J. et al. (2006) *Bioinformatics* **22**, 2695–2696.

'BLAST' is the work of Altschul et al.: Altschul, S.F. et al. (1990) *J. Mol. Biol.* **215**, 403–410.

Full details of the 'BLAST' algorithm, along with download and installation instructions can be obtained from:
http://www.ncbi.nlm.nih.gov/BLAST/.

See Also

plot.blast, hmmer, seqaln

Examples

```
## Not run:
pdb <- read.pdb("1bg2")
blast <- blast.pdb( pdbseq(pdb) )

head(blast$hit.tbl)
top.hits <- plot(blast)
head(top.hits$hits)

## Use 'get.blast()' to retrieve results at a later time.
x <- get.blast(blast$url)
head(x$hit.tbl)

## End(Not run)
```

---

bounds *Bounds of a Numeric Vector*

Description

Find the 'bounds' (i.e. start, end and length) of consecutive numbers within a larger set of numbers in a given vector.

Usage

```
bounds(nums, dup.inds=FALSE, pre.sort=TRUE)
```

Arguments

nums	a numeric vector.
dup.inds	logical, if TRUE the bounds of consecutive duplicated elements are returned.
pre.sort	logical, if TRUE the input vector is ordered prior to bounds determination.

Details

This is a simple utility function useful for summarizing the contents of a numeric vector. For example: find the start position, end position and lengths of secondary structure elements given a vector of residue numbers obtained from a DSSP secondary structure prediction.

By setting 'dup.inds' to TRUE then the indices of the first (start) and last (end) duplicated elements of the vector are returned. For example: find the indices of atoms belonging to a particular residue given a vector of residue numbers (see below).

Value

Returns a three column matrix listing starts, ends and lengths.

Author(s)

Barry Grant

References

Grant, B.J. et al. (2006) *Bioinformatics* **22**, 2695–2696.

Examples

```
test <- c(seq(1,5,1),8,seq(10,15,1))
bounds(test)

test <- rep(c(1,2,4), times=c(2,3,4))
bounds(test, dup.ind=TRUE)
```

`bwr.colors` *Color Palettes*

Description

Create a vector of 'n' "contiguous" colors forming either a Blue-White-Red or a White-Gray-Black color palette.

Usage

```
bwr.colors(n)
mono.colors(n)
```

Arguments

`n`	the number of colors in the palette (>=1).

Details

The function `bwr.colors` returns a vector of `n` color names that range from blue through white to red.

The function `mono.colors` returns color names ranging from white to black. Note: the first element of the returned vector will be NA.

Value

Returns a character vector, `cv`, of color names. This can be used either to create a user-defined color palette for subsequent graphics with `palette(cv)`, or as a `col=` specification in graphics functions and `par`.

Author(s)

Barry Grant

References

Grant, B.J. et al. (2006) *Bioinformatics* **22**, 2695–2696.

The `bwr.colors` function is derived from the `gplots` package function `colorpanel` by Gregory R. Warnes.

See Also

`vmd.colors`, `cm.colors`, `colors`, `palette`, `hsv`, `rgb`, `gray`, `col2rgb`

Examples

```
# Color a distance matrix
pdb <- read.pdb( "1bg2" )
d <- dm(pdb,"calpha")

plot(d, color.palette=bwr.colors)

plot(d,
     resnum.1 = pdb$atom[pdb$calpha,"resno"],
     color.palette = mono.colors,
     xlab="Residue Number", ylab="Residue Number")
```

---

chain.pdb *Find Possible PDB Chain Breaks*

Description

Find possible chain breaks based on connective Calpha atom separation.

Usage

```
chain.pdb(pdb, ca.dist = 4, blank = "X", rtn.vec = TRUE)
```

Arguments

`pdb`	a PDB structure object obtained from `read.pdb`.
`ca.dist`	the maximum distance that separates Calpha atoms considered to be in the same chain.
`blank`	a character to assign non-protein atoms.
`rtn.vec`	logical, if TRUE then the one-letter chain vector consisting of the 26 upper-case letters of the Roman alphabet is returned.

Details

This is a basic function for finding possible chain breaks in PDB structure files, i.e. connective Calpha atoms that are further than `ca.dist` apart.

Value

Prints basic chain information and if `rtn.vec` is TRUE returns a character vector of chain ids consisting of the 26 upper-case letters of the Roman alphabet plus possible `blank` entries for non-protein atoms.

Author(s)

Barry Grant

References

Grant, B.J. et al. (2006) *Bioinformatics* **22**, 2695–2696.

See Also

`read.pdb`, `atom.select`, `trim.pdb`, `write.pdb`

Examples

```
full.pdb <- read.pdb( get.pdb("5p21", URLonly=TRUE) )
inds <- atom.select(full.pdb, resno=c(10:20,30:33))
cut.pdb <- trim.pdb(full.pdb, inds)
chain.pdb(cut.pdb)
```

---

check.utility *Check on Missing Utility Programs*

Description

Internally used in examples, tests, or vignettes.

Usage

```
check.utility(x = c("muscle", "dssp", "stride", "mustang", "makeup"), quiet = TRUE)
```

Arguments

`x`	Names of one or more utility programs to check.
`quiet`	logical, if TRUE no warning or message printed.

Details

Check if requested utility programs are availabe or not.

Value

logical, TRUE if programs are available and FALSE if any one of them is missing.

Examples

```
check.utility(c("muscle", "dssp"), quiet=FALSE)
if(!check.utility("mustang"))
   cat(" The utility program, MUSTANG, is missing on your system\n")
```

---

cmap *Contact Map*

Description

Construct a Contact Map for Given Protein Structure(s).

Usage

```
cmap(xyz, grpby=NULL, dcut = 4, scut = 3, pcut=1, mask.lower = TRUE,
         ncore=1, nseg.scale=1)
```

Arguments

xyz	numeric vector of xyz coordinates or a numeric matrix of coordinates with a row per structure/frame.
grpby	a vector counting connective duplicated elements that indicate the elements of xyz that should be considered as a group (e.g. atoms from a particular residue).
dcut	a cuttoff distance value below which atoms are considered in contact.
scut	a cuttoff neighbour value which has the effect of excluding atoms that are sequentially within this value.
pcut	a cutoff probability of structures/frames showing a contact, above which atoms are considered in contact with respect to the ensemble
mask.lower	logical, if TRUE the lower matrix elements (i.e. those below the diagonal) are returned as NA.
ncore	number of CPU cores used to do the calculation. ncore>1 requires package 'parallel' installed.
nseg.scale	split input data into specified number of segments prior to running multiple core calculation. See fit.xyz.

Details

A contact map is a simplified distance matrix. See the distance matrix function dm for further details.

Value

Returns a N by N numeric matrix composed of zeros and ones, where one indicates a contact between selected atoms.

Author(s)

Barry Grant

References

Grant, B.J. et al. (2006) *Bioinformatics* **22**, 2695–2696.

See Also

dm, dccm, dist, dist.xyz

Examples

```
##- Read PDB file
pdb <- read.pdb( system.file("examples/hivp.pdb", package="bio3d") )

## Atom Selection indices
inds <- atom.select(pdb, "calpha")

## Reference contact map
ref.cont <- cmap( pdb$xyz[inds$xyz], dcut=6, scut=3 )
plot.dmat(ref.cont)

## Not run:
##- Read Traj file
trj <- read.dcd( system.file("examples/hivp.dcd", package="bio3d") )
## For each frame of trajectory
sum.cont <- NULL
for(i in 1:nrow(trj)) {

  ## Contact map for frame 'i'
  cont <- cmap(trj[i,inds$xyz], dcut=6, scut=3)

  ## Product with reference
  prod.cont <- ref.cont * cont
  sum.cont <- c(sum.cont, sum(prod.cont,na.rm=TRUE))
}

plot(sum.cont, typ="l")

## End(Not run)
```

`cmap.filter` *Contact Map Consensus Filtering*

Description

This function filters a tridimensional contact matrix (nXnXz, where n is the residue number and z is the simulation number) selecting only the contact present in at least p simulations, where p<=z.

Usage

```
cmap.filter(cm, cutoff.sims = dim(cm)[3])
```

Arguments

`cm`	A numeric array with 3 dimensions (nXnXz) containing binary contact values. "n" is the residue number, "z" the simulation number. The matrix elements should be 1 if two residues are in contact and 0 if they are not in contact.
`cutoff.sims`	A single element numeric vector corresponding to the minimum number of simulations a contact between two residues must be present. If not, it will be set to 0 in the output matrix.

Value

The output matrix is a nXn binary matrix (n = residue number). Elements equal to 1 correspond to residues in contact, elements equal to 0 to residues not in contact.

See Also

cmap (require bio3d package)

Examples

```
## Not run:
  ## need abind package
  if(!require(abind)) {
     install.packages("abind")
     require(abind)
  }

  ## load example data
  pdbfile <- system.file("examples/hivp.pdb", package="bio3d")
  pdb <- read.pdb(pdbfile)

  trtfile <- system.file("examples/hivp.dcd", package="bio3d")
  trj <- read.dcd(trtfile, verbose=FALSE)

  ## split the trj example in two
  num.of.frames <- dim(trj)[1]
```

```
  trj1 <- trj[1:(num.of.frames/2),]
  trj2 <- trj[((num.of.frames/2)+1):num.of.frames,]

  ## Lets work with Calpha atoms only
  ca.inds <- atom.select(pdb, "calpha")
  #noh.inds <- atom.select(pdb, "noh")

  ## calculate single contact map matrices
  cm.1 <- cmap(trj1[,ca.inds$xyz], pcut=0.3, scut=0, dcut=7, mask.lower=FALSE)
  cm.2 <- cmap(trj2[,ca.inds$xyz], pcut=0.3, scut=0, dcut=5, mask.lower=FALSE)

  ## create a 3D contact matrix from 3 simulations
  cm.all <- abind(cm.1, cm.2, along=3)

  ## calculate average contact matrix
  cm.filter <- cmap.filter(cm=cm.all, cutoff.sims=2)

  ## plot the result
  par(pty="s", mfcol=c(1,3))
  image(cm.1, col=c(NA,"black"))
  image(cm.2, col=c(NA,"black"))
  image(cm.filter, col=c(NA,"black"))

## End(Not run)
```

---

cna *Protein Dynamic Correlation Network Construction and Community Analysis.*

Description

This function builds both residue-based and community-based undirected weighted network graphs from an input correlation matrix, as obtained from the functions 'dccm', 'dccm.nma', and 'dccm.enma'. Community detection/clustering is performed on the initial residue based network to determine the community organization and network structure of the community based network.

Usage

```
  cna(cij, ...)
  ## S3 method for class 'dccm'
cna(cij, cutoff.cij=0.4, cm=NULL,  vnames=colnames(cij),
      cluster.method="btwn", collapse.method="max",
      cols=vmd.colors(), minus.log=TRUE, ...)
  ## S3 method for class 'ensmb'
cna(cij, ..., ncore = NULL)
```

Arguments

cij	A numeric array with 2 dimensions (nXn) containing atomic correlation values, where "n" is the residue number. The matrix elements should be in between 0 and 1 (atomic correlations). Can be also a set of correlation matrices for ensemble network analysis. See 'dccm' function in bio3d package for further details.
...	Additional arguments passed to the methods cna.dccm and cna.ensmb.
cutoff.cij	Numeric element specifying the cutoff on cij matrix values. Coupling below cutoff.cij are set to 0.
cm	(optinal) A numeric array with 2 dimensions (nXn) containing binary contact values, where "n" is the residue number. The matrix elements should be 1 if two residues are in contact and 0 if not in contact. See the 'cmap' function in bio3d package for further details.
vnames	A vector of names for each column in the input cij. This will be used for referencing residues in a similar way to residue numbers in later analysis.
cluster.method	A character string specifying the method for community determination. Supported methods are: btwn="Girvan-Newman betweenness" walk="Random walk" greed="Greedy algorithm for modularity optimization"
collapse.method	A single element character vector specifing the 'cij' collapse method, can be one of 'max', 'median', 'mean', or 'trimmed'. By defualt the 'max' method is used to collapse the input residue based 'cij' matrix into a smaller community based network by taking the maximium 'abs(cij)' value between communities as the comunity-to-community cij value for clustered network construction.
cols	A vector of colors assigned to network nodes.
minus.log	Logical, indicating whether '-log(abs(cij))' values should be used for network construction.
ncore	Number of CPU cores used to do the calculation. By default, use all available cores.

Details

The input to this function should be a correlation matrix as obtained from the 'dccm', 'dccm.mean' or 'dccm.nma' and related functions. Optionally, a contact map 'cm' may also given as input to filter the correlation matrix resulting in the exclusion of network edges between non-contacting atom pairs (as defined in the contact map).

Internally this function calls the igraph package functions 'graph.adjacency', 'edge.betweenness.community', 'walktrap.community', 'fastgreedy.community'. The first constructs an undirected weighted network graph. The second performs Girvan-Newman style clustering by calculating the edge betweenness of the graph, removing the edge with the highest edge betweenness score, calculates modularity (i.e. the difference between the current graph partition and the partition of a random graph, see Newman and Girvan, Physical Review E (2004), Vol 69, 026113), then recalculating

edge betweenness of the edges and again removing the one with the highest score, etc. The returned community partition is the one with the highest overall modularity value. 'walktrap.community' implements the Pons and Latapy algorithm based on the idea that random walks on a graph tend to get "trapped" into densely connected parts of it, i.e. a community. The random walk process is used to determine a distance between nodes. Nodes with low distance values are joined in the same community. 'fastgreedy.community' instead determines the community structure based on the optimization of the modularity. In the starting state each node is isolated and belongs to a separated community. Communities are then joined together (according to the network edges) in pairs and the modularity is calculated. At each step the join resulting in the highest increase of modularity is chosen. This process is repeated until a single community is obtained, then the partitioning with the highest modularity score is selected.

Value

Returns a list object that includes igraph network and community objects with the following components:

`network`	An igraph residue-wise graph object. See below for more details.
`communities`	An igraph residue-wise community object. See below for more details.
`communitiy.network`	An igraph community-wise graph object. See below for more details.
`community.cij`	Numeric square matrix containing the absolute values of the atomic correlation input matrix for each community as obtained from 'cij' via application of 'collapse.method'.
`cij`	Numeric square matrix containing the absolute values of the atomic correlation input matrix.

Author(s)

Guido Scarabelli and Barry Grant

See Also

`plot.cna`, `summary.cna`, `view.cna`, `graph.adjacency`, `edge.betweenness.community`, `walktrap.community`, `fastgreedy.community`

Examples

```
##-- Build a correlation network from NMA results
## Read example PDB
pdb <- read.pdb("4Q21")

## Perform NMA
modes <- nma(pdb)
#plot(modes, sse=pdb)

## Calculate correlations
cij <- dccm(modes)
```

```
#plot(cij, sse=pdb)

## Build, and betweenness cluster, a network graph
net <- cna(cij, cutoff.cij=0.35)
#plot(net, pdb)

## within VMD set 'coloring method' to 'Chain' and 'Drawing method' to Tube
#view.cna(net, trim.pdb(pdb, atom.select(pdb,"calpha")), launch=TRUE )

##-- Build a correlation network from MD results
## Read example trajectory file
trtfile <- system.file("examples/hivp.dcd", package="bio3d")
trj <- read.dcd(trtfile)

## Read the starting PDB file to determine atom correspondence
pdbfile <- system.file("examples/hivp.pdb", package="bio3d")
pdb <- read.pdb(pdbfile)

## select residues 24 to 27 and 85 to 90 in both chains
inds <- atom.select(pdb,"///24:27,85:90///CA/")

## lsq fit of trj on pdb
xyz <- fit.xyz(pdb$xyz, trj, fixed.inds=inds$xyz, mobile.inds=inds$xyz)

## calculate dynamical cross-correlation matrix
cij <- dccm(xyz)

## Build, and betweenness cluster, a network graph
net <- cna(cij)

# Plot coarse grained network based on dynamically coupled communities
xy <- plot.cna(net)
plot.dccm2(cij, margin.segments=net$communities$membership)


##-- Begin to examine network structure - see CNA vignette for more details
net
summary(net)
attributes(net)
table( net$communities$members )
```

---

com *Center of Mass*

Description

Calculate the center of mass of a PDB object.

Usage

```
com(pdb, inds=NULL, use.mass=TRUE, ...)
com.xyz(xyz, mass=NULL)
```

Arguments

pdb	an object of class pdb as obtained from function read.pdb.
inds	atom and xyz coordinate indices obtained from atom.select that selects the elements of pdb upon which the calculation should be based.
use.mass	logical, if TRUE the calculation will be mass weighted (center of mass).
...	additional arguments to atom2mass.
xyz	a numeric vector of Cartesian coordinates.
mass	a numeric vector containing the masses of each atom in xyz.

Details

This function calculates the center of mass of the provided PDB structure. Atom names found in standard amino acids in the PDB are mapped to atom elements and their corresponding relative atomic masses.

In the case of an unknown atom name elety.custom and mass.custom can be used to map an atom to the correct atomic mass. See examples for more details.

Alternatively, the atom name will be mapped automatically to the element corresponding to the first character of the atom name. Atom names starting with character H will be mapped to hydrogen atoms.

Value

Returns the Cartesian coordinates at the center of mass.

Author(s)

Lars Skjaerven

References

Grant, B.J. et al. (2006) *Bioinformatics* **22**, 2695–2696.

See Also

atom2mass

Examples

```
## Stucture of PKA:
pdb <- read.pdb("3dnd")

## Center of mass:
com(pdb)

## Center of mass of a selection
inds <- atom.select(pdb, chain="I")
com(pdb, inds)



## Not run:
## Unknown atom names
pdb <- read.pdb("3dnd")
inds <- atom.select(pdb, resid="LL2")
mycom <- com(pdb, inds, rescue=TRUE)
#warnings()


## Map atom names manually
pdb <- read.pdb("3RE0")
inds <- atom.select(pdb, resno=201)

elety.cust <- list("CL2"="Cl", "PT1"="Pt")
mass.cust <- list("Cl"=35.45, "Pt"=195.08)

mycom <- com(pdb, inds, elety.custom=elety.cust, mass.custom=mass.cust,
             rescue=TRUE)

## End(Not run)
```

`combine.sel` *Combine Atom Selections From PDB Structure*

Description

Do "and", "or", or "not" logical operations between two atom selections made by `atom.select`

Usage

```
combine.sel(sel1=NULL, sel2=NULL, op="AND", verbose=TRUE)
```

Arguments

`sel1`	an atom selection object of class `"select"`, obtained from `atom.select`.

sel2	a second atom selection object of class "select", obtained from atom.select.
op	name of the logical operation.
verbose	logical, if TRUE details of the selection combination are printed.

Details

The value of op should be one of following: (1) "AND", "and", or "&" for set intersection, (2) "OR", "or", "|", or "+" for set union, (3) "NOT", "not", "!", or "-" for set difference sel1 - sel2.

Value

Returns a list of class "select" with components:

atom	atom indices of selected atoms.
xyz	xyz indices of selected atoms.

Author(s)

Xin-Qiu Yao

References

Grant, B.J. et al. (2006) *Bioinformatics* **22**, 2695–2696.

See Also

atom.select

Examples

```
# Read a PDB file
pdb <- read.pdb( system.file("examples/1hel.pdb", package="bio3d") )

# Select all C-alpha atoms
ca.inds <- atom.select(pdb, "calpha")

# Select all atoms with residues numbers between 43 and 54
res.inds <- atom.select(pdb, resno=43:54)

# Select all C-beta atoms with residues numbers between 43 and 54
cb.inds <- atom.select(pdb, resno=43:54, elety="CB")

# Intersection
inds <- combine.sel(ca.inds, res.inds, op="AND")
print( pdb$atom[ inds$atom, "resid" ] )
print( pdb$xyz[ inds$xyz ] )

# Union
inds2 <- combine.sel(inds, cb.inds, op="+")
```

```
# Not
inds3 <- combine.sel(res.inds, ca.inds, op="-")
```

`community.tree` *Reconstruction of the Girvan-Newman Community Tree for a CNA Class Object.*

Description

This function reconstructs the community tree of the community clustering analysis performed by the 'cna' function. It allows the user to explore different network community partitions.

Usage

```
community.tree(x, rescale=FALSE)
```

Arguments

`x`	A protein network graph object as obtained from the 'cna' function.
`rescale`	Logical, indicating whether to rescale the community names starting from 1. If FALSE, the community names will start from N+1, where N is the number of nodes.

Details

The input of this function should be a 'cna' class object containing 'network' and 'communities' attributes.

This function reconstructs the community residue memberships for each modularity value. The purpose is to facilitate inspection of alternate community partitioning points, which in practice often corresponds to a value close to the maximum of the modularity, but not the maximum value itself.

Value

Returns a list object that includes the following components:

`modularity`	A numeric vector containing the modularity values.
`tree`	A numeric matrix containing in each row the community residue memberships corresponding to a modularity value. The rows are ordered according to the 'modularity' object.
`num.of.comms`	A numeric vector containing the number of communities per modularity value. The vector elements are ordered according to the 'modularity' object.

Author(s)

Guido Scarabelli

See Also

cna, network.amendment, summary.cna

Examples

```
###-- Build a CNA object
pdb <- read.pdb("4Q21")
modes <- nma(pdb)
cij <- dccm(modes)
net <- cna(cij, cutoff.cij=0.2)


##-- Reconstruct the community membership vector for each clustering step.
tree <- community.tree(net, rescale=TRUE)

## Plot modularity vs number of communities
plot( tree$num.of.comms, tree$modularity )

## Inspect the maximum modularity value partitioning
max.mod.ind <- which.max(tree$modularity)

## Number of communities (k) at max modularity
tree$num.of.comms[ max.mod.ind ]

## Membership vector at this partition point
tree$tree[max.mod.ind,]

# Should be the same as that contained in the original CNA network object
net$communities$membership == tree$tree[max.mod.ind,]

# Inspect a new membership partitioning (at k=7)
memb.k7 <- tree$tree[ tree$num.of.comms == 7, ]

## Produce a new k=7 community network
net.7 <- network.amendment(net, memb.k7)
plot(net.7, pdb)
#view.cna(net.7, trim.pdb(pdb, atom.select(pdb,"calpha")), launch=TRUE )
```

---

consensus *Sequence Consensus for an Alignment*

Description

Determines the consensus sequence for a given alignment at a given identity cutoff value.

Usage

```
consensus(alignment, cutoff = 0.6)
```

Arguments

alignment	an alignment object created by the read.fasta function or an alignment character matrix.
cutoff	a numeric value beteen 0 and 1, indicating the minimum sequence identity threshold for determining a consensus amino acid. Default is 0.6, or 60 percent residue identity.

Value

A vector containing the consensus sequence, where '-' represents positions with no consensus (i.e. under the cutoff)

Author(s)

Barry Grant

References

Grant, B.J. et al. (2006) *Bioinformatics* **22**, 2695–2696.

See Also

read.fasta

Examples

```
#-- Read HIV protease alignment
aln <- read.fasta(system.file("examples/hivp_xray.fa",package="bio3d"))

# Generate consensus
con <- consensus(aln)
print(con$seq)

# Plot residue frequency matrix
##png(filename = "freq.png", width = 1500, height = 780)
col <- mono.colors(32)
aa  <- rev(rownames(con$freq))

image(x=1:ncol(con$freq),
      y=1:nrow(con$freq),
      z=as.matrix(rev(as.data.frame(t(con$freq)))),
      col=col, yaxt="n", xaxt="n",
      xlab="Alignment Position", ylab="Residue Type")

# Add consensus along the axis
axis(side=1, at=seq(0,length(con$seq),by=5))
```

```
axis(side=2, at=c(1:22), labels=aa)
axis(side=3, at=c(1:length(con$seq)), labels =con$seq)
axis(side=4, at=c(1:22), labels=aa)
grid(length(con$seq), length(aa))
box()

# Add consensus sequence
for(i in 1:length(con$seq)) {
  text(i, which(aa==con$seq[i]),con$seq[i],col="white")
}

# Add lines for residue type separation
abline(h=c(2.5,3.5, 4.5, 5.5, 3.5, 7.5, 9.5,
         12.5, 14.5, 16.5, 19.5), col="gray")
```

`conserv` *Score Residue Conservation At Each Position in an Alignment*

Description

Quantifies residue conservation in a given protein sequence alignment by calculating the degree of amino acid variability in each column of the alignment.

Usage

```
conserv(x, method = c("similarity","identity","entropy22","entropy10"),
        sub.matrix = c("bio3d", "blosum62", "pam30", "other"),
        matrix.file = NULL, normalize.matrix = TRUE)
```

Arguments

`x`	an alignment list object with `id` and `ali` components, similar to that generated by `read.fasta`.
`method`	the conservation assesment method.
`sub.matrix`	a matrix to score conservation.
`matrix.file`	a file name of an arbitary user matrix.
`normalize.matrix`	logical, if TRUE the matrix is normalized pior to assesing conservation.

Details

To assess the level of sequence conservation at each position in an alignment, the "similarity", "identity", and "entropy" per position can be calculated.

The "similarity" is defined as the average of the similarity scores of all pairwise residue comparisons for that position in the alignment, where the similarity score between any two residues is the score value between those residues in the chosen substitution matrix "sub.matrix".

The "identity" i.e. the preference for a specific amino acid to be found at a certain position, is assessed by averaging the identity scores resulting from all possible pairwise comparisons at that position in the alignment, where all identical residue comparisons are given a score of 1 and all other comparisons are given a value of 0.

"Entropy" is based on Shannons information entropy. See the `entropy` function for further details.

Note that the returned scores are normalized so that conserved columns score 1 and diverse columns score 0.

Value

Returns a numeric vector of scores

Note

Each of these conservation scores has particular strengths and weaknesses. For example, entropy elegantly captures amino acid diversity but fails to account for stereochemical similarities. By employing a combination of scores and taking the union of their respective conservation signals we expect to achieve a more comprehensive analysis of sequence conservation (Grant, 2007).

Author(s)

Barry Grant

References

Grant, B.J. et al. (2006) *Bioinformatics* **22**, 2695–2696. Grant, B.J. et al. (2007) *J. Mol. Biol.* **368**, 1231–1248.

See Also

`read.fasta`, `read.fasta.pdb`

Examples

```
## Read an example alignment
aln <- read.fasta(system.file("examples/hivp_xray.fa",package="bio3d"))

## Score conservation
conserv(x=aln$ali, method="similarity", sub.matrix="bio3d")
##conserv(x=aln$ali,method="entropy22", sub.matrix="other")
```

convert.pdb *Renumber and Convert Between Various PDB formats*

Description

Renumber and convert between CHARMM, Amber, Gromacs and Brookhaven PDB formats.

Usage

```
convert.pdb(pdb, type="original",
                renumber = FALSE, first.resno = 1, first.eleno = 1,
                consecutive=TRUE, rm.h = TRUE, rm.wat = FALSE,
                verbose=TRUE)
```

Arguments

pdb	a structure object of class "pdb", obtained from read.pdb.
type	output format, one of 'original', 'pdb', 'charmm', 'amber', or 'gromacs'. The default option of 'original' results in no conversion.
renumber	logical, if TRUE atom and residue records are renumbered using 'first.resno' and 'first.eleno'.
first.resno	first residue number to be used if 'renumber' is TRUE.
first.eleno	first element number to be used if 'renumber' is TRUE.
consecutive	logical, if TRUE renumbering will result in consecutive residue numbers spanning all chains. Otherwise new residue numbers will begin at 'first.resno' for each chain.
rm.h	logical, if TRUE hydrogen atoms are removed.
rm.wat	logical, if TRUE water atoms are removed.
verbose	logical, if TRUE details of the conversion process are printed.

Details

Convert atom names and residue names, renumber atom and residue records, strip water and hydrogen atoms from pdb objects.

Format type can be one of "ori", "pdb", "charmm", "amber" or "gromacs".

Value

Returns a list of class "pdb", with the following components:

atom	a character matrix containing all atomic coordinate ATOM data, with a row per ATOM and a column per record type. See below for details of the record type naming convention (useful for accessing columns).
het	a character matrix containing atomic coordinate records for atoms within "non-standard" HET groups (see atom).

helix	'start', 'end' and 'length' of H type sse, where start and end are residue numbers "resno".
sheet	'start', 'end' and 'length' of E type sse, where start and end are residue numbers "resno".
seqres	sequence from SEQRES field.
xyz	a numeric vector of ATOM coordinate data.
calpha	logical vector with length equal to `nrow(atom)` with TRUE values indicating a C-alpha "elety".

Note

For both `atom` and `het` list components the column names can be used as a convenient means of data access, namely: Atom serial number "eleno" , Atom type "elety", Alternate location indicator "alt", Residue name "resid", Chain identifier "chain", Residue sequence number "resno", Code for insertion of residues "insert", Orthogonal coordinates "x", Orthogonal coordinates "y", Orthogonal coordinates "z", Occupancy "o", and Temperature factor "b". See examples for further details.

Author(s)

Barry Grant

References

Grant, B.J. et al. (2006) *Bioinformatics* **22**, 2695–2696.

For a description of PDB format (version3.3) see:
http://www.wwpdb.org/documentation/format33/v3.3.html.

See Also

`atom.select`, `write.pdb`, `read.dcd`, `read.fasta.pdb`, `read.fasta`

Examples

```
# Read a PDB file
pdb <- read.pdb("4q21")
pdb
head( pdb$atom[pdb$calpha,"resno"] )

# Convert to CHARMM format
new <- convert.pdb(pdb, type="amber", renumber=TRUE, first.resno=22 )
head( new$atom[new$calpha,"resno"] )

# Write a PDB file
#write.pdb(new, file="tmp4amber.pdb")
```

core.find *Identification of Invariant Core Positions*

Description

Perform iterated rounds of structural superposition to identify the most invariant region in an aligned set of protein structures.

Usage

```
core.find(aln, shortcut = FALSE, rm.island = FALSE,
          verbose = TRUE, stop.at = 15, stop.vol = 0.5,
          write.pdbs = FALSE, outpath="core_pruned",
          ncore = 1, nseg.scale = 1)
```

Arguments

aln	a numeric matrix of aligned C-alpha xyz Cartesian coordinates. For example an alignment data structure obtained with read.fasta.pdb or a trajectory subset obtained from read.dcd.
shortcut	if TRUE, remove more than one position at a time.
rm.island	remove isolated fragments of less than three residues.
verbose	logical, if TRUE a "core_pruned" directory containing 'core structures' for each iteraction is written to the current directory.
stop.at	minimal core size at which iterations should be stopped.
stop.vol	minimal core volume at which iterations should be stopped.
write.pdbs	logical, if TRUE core coordinate files, containing only core positions for each iteration, are written to a location specified by outpath.
outpath	character string specifying the output directory when write.pdbs is TRUE.
ncore	number of CPU cores used to do the calculation. ncore>1 requires package 'parallel' installed.
nseg.scale	split input data into specified number of segments prior to running multiple core calculation. See fit.xyz.

Details

This function attempts to iteratively refine an initial structural superposition determined from a multiple alignment. This involves iterated rounds of superposition, where at each round the position(s) displaying the largest differences is(are) excluded from the dataset. The spatial variation at each aligned position is determined from the eigenvalues of their Cartesian coordinates (i.e. the variance of the distribution along its three principal directions). Inspired by the work of Gerstein *et al.* (1991, 1995), an ellipsoid of variance is determined from the eigenvalues, and its volume is taken as a measure of structural variation at a given position.

Optional "core PDB files" containing core positions, upon which superposition is based, can be written to a location specified by outpath by setting write.pdbs=TRUE. These files are useful for examining the core filtering process by visualising them in a graphics program.

Value

Returns a list of class "core" with the following components:

volume	total core volume at each fitting iteration/round.
length	core length at each round.
resno	residue number of core residues at each round (taken from the first aligned structure) or, alternatively, the numeric index of core residues at each round.
step.inds	atom indices of core atoms at each round.
atom	atom indices of core positions in the last round.
xyz	xyz indices of core positions in the last round.
c1A.atom	atom indices of core positions with a total volume under 1 Angstrom\^3.
c1A.xyz	xyz indices of core positions with a total volume under 1 Angstrom\^3.
c1A.resno	residue numbers of core positions with a total volume under 1 Angstrom\^3.
c0.5A.atom	atom indices of core positions with a total volume under 0.5 Angstrom\^3.
c0.5A.xyz	xyz indices of core positions with a total volume under 0.5 Angstrom\^3.
c0.5A.resno	residue numbers of core positions with a total volume under 0.5 Angstrom\^3.

Note

The relevance of the 'core positions' identified by this procedure is dependent upon the number of input structures and their diversity.

Author(s)

Barry Grant

References

Grant, B.J. et al. (2006) *Bioinformatics* **22**, 2695–2696.

Gerstein and Altman (1995) *J. Mol. Biol.* **251**, 161–175.

Gerstein and Chothia (1991) *J. Mol. Biol.* **220**, 133–149.

See Also

read.fasta.pdb, plot.core, fit.xyz

Examples

```
## Not run:
##-- Generate a small kinesin alignment and read corresponding structures
pdbfiles <- get.pdb(c("1bg2","2ncd","1i6i","1i5s"), URLonly=TRUE)
pdbs <- pdbaln(pdbfiles)

##-- Find 'core' positions
core <- core.find(pdbs)
plot(core)
```

```
##-- Fit on these relatively invarient subset of positions
#core.inds <- print(core, vol=1)
core.inds <- print(core, vol=0.5)
xyz <- pdbfit(pdbs, core.inds, outpath="corefit_structures")

##-- Compare to fitting on all equivalent positions
xyz2 <- pdbfit(pdbs)

## Note that overall RMSD will be higher but RMSF will
##  be lower in core regions, which may equate to a
##  'better fit' for certain applications
gaps <- gap.inspect(pdbs$xyz)
rmsd(xyz[,gaps$f.inds])
rmsd(xyz2[,gaps$f.inds])

plot(rmsf(xyz[,gaps$f.inds]), typ="l", col="blue", ylim=c(0,9))
points(rmsf(xyz2[,gaps$f.inds]), typ="l", col="red")

## End(Not run)

## Not run:
##-- Run core.find() on a trajectory
trtfile <- system.file("examples/hivp.dcd", package="bio3d")
trj <- read.dcd(trtfile)

## Read the starting PDB file to determine atom correspondence
pdbfile <- system.file("examples/hivp.pdb", package="bio3d")
pdb <- read.pdb(pdbfile)

## select calpha coords from a manageable number of frames
ca.ind <- atom.select(pdb, "calpha")$xyz
frames <- seq(1, nrow(trj), by=10)

core <- core.find( trj[frames, ca.ind], write.pdbs=TRUE )

## have a look at the various cores "vmd -m core_pruned/*.pdb"

## Lets use a 6A^3 core cutoff
inds <- print(core, vol=6)
write.pdb(xyz=pdb$xyz[inds$xyz],resno=pdb$atom[inds$atom,"resno"], file="core.pdb")


##- Fit trj onto starting structure based on core indices
xyz <- fit.xyz( fixed = pdb$xyz,
                mobile = trj,
                fixed.inds  = inds$xyz,
                mobile.inds = inds$xyz)

##write.pdb(pdb=pdb, xyz=xyz, file="new_trj.pdb")
##write.ncdf(xyz, "new_trj.nc")
```

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

`cov.nma` *Calculate Covariance Matrix from Normal Modes*

Description

Calculate the covariance matrix from a normal mode object.

Usage

```
## S3 method for class 'nma'
cov(nma)
## S3 method for class 'enma'
cov(enma, ncore=NULL)
```

Arguments

`nma`	an `nma` object as obtained from function `nma.pdb`.
`enma`	an `enma` object as obtained from function `nma.pdbs`.
`ncore`	number of CPU cores used to do the calculation. `ncore>1` requires package 'parallel' installed.

Details

This function calculates the covariance matrix from a `nma` object as obtained from function `nma.pdb` or covariance matrices from a `enma` object as obtain from function `nma.pdbs`.

Value

Returns the calculated covariance matrix (function `cov.nma`), or covariance matrices (function `cov.enma`).

Author(s)

Lars Skjaerven

References

Grant, B.J. et al. (2006) *Bioinformatics* **22**, 2695–2696. Fuglebakk, E. et al. (2013) *JCTC* **9**, 5618–5628.

See Also

`nma`

`covsoverlap` *Covariance Overlap*

Description

Calculate the ovariance overlap obtained from NMA.

Usage

```
covsoverlap(...)
## S3 method for class 'enma'
covsoverlap(enma, ncore=NULL, ...)
## S3 method for class 'nma'
covsoverlap(a, b, subset=NULL, ...)
```

Arguments

`enma`	an object of class `"enma"` obtained from function `nma.pdbs`.
`ncore`	number of CPU cores used to do the calculation. `ncore>1` requires package 'parallel' installed.
`a`	a list object with elements 'U' and 'L' (e.g. as obtained from function `nma`) containing the eigenvectors and eigenvalues, respectively, to be compared with `b`.
`b`	a list object with elements 'U' and 'L' (e.g. as obtained from function `nma`) containing the eigenvectors and eigenvalues, respectively, to be compared with `a`.
`subset`	the number of modes to consider.
`...`	arguments passed to associated functions.

Details

Covariance overlap is a measure for the similarity between two covariance matrices, e.g. obtained from NMA.

Value

Returns the similarity coefficient(s).

Author(s)

Lars Skjaerven

References

Grant, B.J. et al. (2006) *Bioinformatics* **22**, 2695–2696. Romo, T.D. et al. (2011) *Proteins* **79**, 23–34.

See Also

Other similarity measures: sip, covsoverlap, bhattacharyya.

---

dccm *DCCM: Dynamical Cross-Correlation Matrix*

Description

Determine the cross-correlations of atomic displacements.

Usage

```
dccm(x, ...)
```

Arguments

x	a numeric matrix of Cartesian coordinates with a row per structure/frame which will br passed to dccm.xyz(). Alternatively, an object of class nma as obtained from function nma that will be passed to the dccm.nma() function, see below for examples.
...	additional arguments passed to the methods dccm.xyz, dccm.pca, dccm.nma, and dccm.enma.

Details

dccm is a generic function calling the corresponding function determined by the class of the input argument x. Use methods("dccm") to get all the methods for dccm generic:

dccm.xyz will be used when x is a numeric matrix containing Cartesian coordinates (e.g. trajectory data).

dccm.pca will calculate the cross-correlations based on an pca object.

dccm.nma will calculate the cross-correlations based on an nma object. Similarly, dccm.enma will calculate the correlation matrices based on an ensemble of nma objects (as obtained from function nma.pdbs).

plot.dccm and view.dccm provides convenient functionality to plot a correlation map, and visualize the correlations in the structure, respectively.

See examples for each corresponding function for more details.

Author(s)

Barry Grant, Lars Skjaerven

References

Grant, B.J. et al. (2006) *Bioinformatics* **22**, 2695–2696.

See Also

dccm.xyz, dccm.nma, dccm.enma, dccm.pca, plot.dccm, view.dccm.

---

dccm.enma *Cross-Correlation for Ensemble NMA (eNMA)*

Description

Calculate the cross-correlation matrices from an ensemble of NMA objects.

Usage

```
## S3 method for class 'enma'
dccm(x, ncore = NULL, na.rm=FALSE, ...)
```

Arguments

x	an object of class enma as obtained from function nma.pdbs.
ncore	number of CPU cores used to do the calculation. ncore>1 requires package 'parallel' installed.
na.rm	logical, if FALSE the DCCM might containt NA values (applies only when the enma object is calculated with argument 'rm.gaps=FALSE').
...	additional arguments passed to dccm.nma.

Details

This is a wrapper function for calling dccm.nma on a collection of 'nma' objects as obtained from function nma.pdbs.

See examples for more details.

Value

Returns a list with the following components:

all.dccm	an array or list containing the correlation matrices for each 'nma' object. An array is returned when the 'enma' object is calculated with 'rm.gaps=TRUE', and a list is used when 'rm.gaps=FALSE'.
avg.dccm	a numeric matrix containing the average correlation matrix. The average is only calculated when the 'enma' object is calculated with 'rm.gaps=TRUE'.

Author(s)

Lars Skjaerven

References

Wynsberghe. A.W.V, Cui, Q. *Structure* **14**, 1647–1653. Grant, B.J. et al. (2006) *Bioinformatics* **22**, 2695–2696.

See Also

nma, dccm.nma, plot.dccm

Examples

```
## NOTE: FOLLOWING EXAMPLE NEEDS MUSCLE INSTALLED
if(check.utility("muscle")) {

## Fetch PDB files and split to chain A only PDB files
ids <- c("1a70_A", "1czp_A", "1frd_A", "1fxi_A", "1iue_A", "1pfd_A")
files <- get.pdb(ids, split = TRUE, path = tempdir())

## Sequence/Structure Alignement
pdbs <- pdbaln(files, outfile = tempfile())

## Normal mode analysis on aligned data
modes <- nma(pdbs)

## Calculate all 6 correlation matrices
cij <- dccm(modes)

## Plot correlations for first structure
plot.dccm(cij$all.dccm[,,1])

}
```

---

dccm.nma *Dynamic Cross-Correlation from Normal Modes Analysis*

Description

Calculate the cross-correlation matrix from Normal Modes Analysis.

Usage

```
## S3 method for class 'nma'
dccm(x, nmodes = NULL, ncore = NULL, ...)
```

Arguments

x	an object of class nma as obtained from function nma.
nmodes	numerical, number of modes to consider.
ncore	number of CPU cores used to do the calculation. ncore>1 requires package 'parallel' installed.
...	additional arguments ?

Details

This function calculates the cross-correlation matrix from Normal Modes Analysis (NMA) obtained from nma of a protein structure. It returns a matrix of residue-wise cross-correlations whose elements, Cij, may be displayed in a graphical representation frequently termed a dynamical cross-correlation map, or DCCM.

If Cij = 1 the fluctuations of residues i and j are completely correlated (same period and same phase), if Cij = -1 the fluctuations of residues i and j are completely anticorrelated (same period and opposite phase), and if Cij = 0 the fluctuations of i and j are not correlated.

Value

Returns a cross-correlation matrix.

Author(s)

Lars Skjaerven

References

Wynsberghe. A.W.V, Cui, Q. *Structure* **14**, 1647–1653. Grant, B.J. et al. (2006) *Bioinformatics* **22**, 2695–2696.

See Also

nma, plot.dccm

Examples

```
## Fetch stucture
pdb <- read.pdb( system.file("examples/1hel.pdb", package="bio3d") )

## Calculate normal modes
modes <- nma(pdb)

## Calculate correlation matrix
cm <- dccm.nma(modes)

## Plot correlation map
plot(cm, sse = pdb, contour = FALSE, col.regions = bwr.colors(20),
```

```
        at = seq(-1, 1, 0.1))
```

---

dccm.pca *Dynamic Cross-Correlation from Principal Component Analysis*

Description

Calculate the cross-correlation matrix from principal component analysis (PCA).

Usage

```
## S3 method for class 'pca'
dccm(x, nmodes = NULL, ncore = NULL, ...)
```

Arguments

x	an object of class pca as obtained from function pca.xyz.
nmodes	numerical, number of modes to consider.
ncore	number of CPU cores used to do the calculation. By default (ncore = NULL), use all available cores detected.
...	additional arguments to cov2dccm.

Details

This function calculates the cross-correlation matrix from principal component analysis (PCA) obtained from pca.xyz of a set of protein structures. It is an alternative way to calculate correlation in addition to the conventional way from xyz coordinates directly. But, in this new way one can freely chooses the number of modes to be included in the calculation (i.e. filter out fast modes).

Value

Returns a cross-correlation matrix.

Author(s)

Xin-Qiu Yao

References

Grant, B.J. et al. (2006) *Bioinformatics* **22**, 2695–2696.

See Also

pca.xyz, plot.dccm

Examples

```
## Not run:
##-- Read example trajectory file
trtfile <- system.file("examples/hivp.dcd", package="bio3d")
trj <- read.dcd(trtfile)

## Read the starting PDB file to determine atom correspondence
pdbfile <- system.file("examples/hivp.pdb", package="bio3d")
pdb <- read.pdb(pdbfile)

## Select residues 24 to 27 and 85 to 90 in both chains
inds <- atom.select(pdb,"///24:27,85:90///CA/")

## lsq fit of trj on pdb
xyz <- fit.xyz(pdb$xyz, trj, fixed.inds=inds$xyz, mobile.inds=inds$xyz)

## Do PCA
pca <- pca.xyz(xyz)

## DCCM
cij <- dccm(pca, nmodes=10)

## Plot DCCM
plot(cij)

## End(Not run)
```

---

dccm.xyz *DCCM: Dynamical Cross-Correlation Matrix*

Description

Determine the cross-correlations of atomic displacements.

Usage

```
## S3 method for class 'xyz'
dccm(x, reference = NULL, grpby=NULL,
ncore=1, nseg.scale=1, ...)
cov2dccm(vcov, method = c("pearson", "lmi"), ncore = NULL)
```

Arguments

x	a numeric matrix of Cartesian coordinates with a row per structure/frame.
reference	The reference structure about which displacements are analysed.
grpby	a vector counting connective duplicated elements that indicate the elements of xyz that should be considered as a group (e.g. atoms from a particular residue).

ncore	number of CPU cores used to do the calculation. ncore>1 requires package 'parallel' installed.
nseg.scale	split input data into specified number of segments prior to running multiple core calculation. See fit.xyz.
...	hmm.
vcov	numeric variance-covariance matrix.
method	method to calculate the cross-correlation.

Details

The extent to which the atomic fluctuations/displacements of a system are correlated with one another can be assessed by examining the magnitude of all pairwise cross-correlation coefficients (see McCammon and Harvey, 1986).

This function returns a matrix of all atom-wise cross-correlations whose elements, Cij, may be displayed in a graphical representation frequently termed a dynamical cross-correlation map, or DCCM.

If Cij = 1 the fluctuations of atoms i and j are completely correlated (same period and same phase), if Cij = -1 the fluctuations of atoms i and j are completely anticorrelated (same period and opposite phase), and if Cij = 0 the fluctuations of i and j are not correlated.

Typical characteristics of DCCMs include a line of strong cross-correlation along the diagonal, cross-correlations emanating from the diagonal, and off-diagonal cross-correlations. The high diagonal values occur where i = j, where Cij is always equal to 1.00. Positive correlations emanating from the diagonal indicate correlations between contiguous residues, typically within a secondary structure element or other tightly packed unit of structure. Typical secondary structure patterns include a triangular pattern for helices and a plume for strands. Off-diagonal positive and negative correlations may indicate potentially interesting correlations between domains of non-contiguous residues.

cov2dccm function calculates the N-by-N cross-correlation matrix directly from a 3N-by-3N variance-covariance matrix. If method = "pearson", the conventional Pearson's inner-product correlaiton calculation will be invoked, in which only the diagnol of each residue-residue covariance sub-matrix is considered.

If method = "lmi", then the linear mutual information cross-correlation will be calculated. 'LMI' considers both diagnol and off-diagnol entries in sub-matrices, and so even grabs the correlation of residues moving on orthognal directions. (See more details in lmi.)

Value

Returns a cross-correlation matrix.

Author(s)

Xin-Qiu Yao and Gisle Saelensminde

References

Grant, B.J. et al. (2006) *Bioinformatics* **22**, 2695–2696.

McCammon, A. J. and Harvey, S. C. (1986) *Dynamics of Proteins and Nucleic Acids*, Cambridge University Press, Cambridge.

Lange, O.F. and Grubmuller, H. (2006) *PROTEINS: Structure, Function, and Bioinformatics* **62**:1053–1061.

See Also

cor for examining xyz cross-correlations, dccm, dccm.nma, dccm.pca, lmi, dccm.enma.

Examples

```
## Not run:
##-- Read example trajectory file
trtfile <- system.file("examples/hivp.dcd", package="bio3d")
trj <- read.dcd(trtfile)

## Read the starting PDB file to determine atom correspondence
pdbfile <- system.file("examples/hivp.pdb", package="bio3d")
pdb <- read.pdb(pdbfile)

## select residues 24 to 27 and 85 to 90 in both chains
inds <- atom.select(pdb,"///24:27,85:90///CA/")

## lsq fit of trj on pdb
xyz <- fit.xyz(pdb$xyz, trj, fixed.inds=inds$xyz, mobile.inds=inds$xyz)

## DCCM (slow to run so restrict to Calpha)
cij <- dccm(xyz)

## Plot DCCM
plot(cij)

## Or
library(lattice)
contourplot(cij, region = TRUE, labels=F, col="gray40",
            at=c(-1, -0.75, -0.5, -0.25, 0.25, 0.5, 0.75, 1),
            xlab="Residue No.", ylab="Residue No.",
            main="DCCM: dynamic cross-correlation map")

## End(Not run)
```

---

deformation.nma *Deformation Analysis*

Description

Calculate deformation energies from Normal Mode Analysis.

Usage

```
deformation.nma(nma, mode.inds = NULL, pfc.fun = NULL, ncore = NULL)
```

Arguments

nma	a list object of class "nma" (obtained with nma).
mode.inds	a numeric vector of mode indices in which the calculation should be based.
pfc.fun	customized pair force constant ('pfc') function. The provided function should take a vector of distances as an argument to return a vector of force constants. See nma for examples.
ncore	number of CPU cores used to do the calculation. ncore>1 requires package 'parallel' installed.

Details

Deformation analysis provides a measure for the amount of local flexibility of the protein structure - i.e. atomic motion relative to neighbouring atoms. It differs from 'fluctuations' (e.g. RMSF values) which provide amplitudes of the absolute atomic motion.

Deformation energies are calculated based on the nma object. By default the first 20 non-trivial modes are included in the calculation.

See examples for more details.

Value

Returns a list with the following components:

ei	numeric matrix containing the energy contribution (E) from each atom (i; row-wise) at each mode index (column-wise).
sums	deformation energies corresponding to each mode.

Author(s)

Lars Skjaerven

References

Hinsen, K. (1998) *Proteins* **33**, 417–429. Grant, B.J. et al. (2006) *Bioinformatics* **22**, 2695–2696.

See Also

nma

Examples

```
## Fetch stucture
pdb <- read.pdb( system.file("examples/1hel.pdb", package="bio3d") )

## Calculate (vibrational) normal modes
modes <- nma(pdb)

## Calculate deformation energies
def.energies <- deformation.nma(modes)


## Not run:
## Fluctuations of first non-trivial mode
def.energies <- deformation.nma(modes, mode.inds=seq(7, 16))

write.pdb(pdb=NULL, xyz=modes$xyz,
          b=def.energies$ei[,1])

## End(Not run)
```

---

diag.ind *Diagonal Indices of a Matrix*

Description

Returns a matrix of logicals the same size of a given matrix with entries 'TRUE' in the upper triangle close to the diagonal.

Usage

```
diag.ind(x, n = 1, diag = TRUE)
```

Arguments

x	a matrix.
n	the number of elements from the diagonal to include.
diag	logical. Should the diagonal be included?

Details

Basic function useful for masking elements close to the diagonal of a given matrix.

Value

Returns a matrix of logicals the same size of a given matrix with entries 'TRUE' in the upper triangle close to the diagonal.

Author(s)

Barry Grant

References

Grant, B.J. et al. (2006) *Bioinformatics* **22**, 2695–2696.

See Also

diag, lower.tri, upper.tri, matrix

Examples

```
diag.ind( matrix(,ncol=5,nrow=5), n=3 )
```

---

difference.vector *Difference Vector*

Description

Define a difference vector between two conformational states.

Usage

```
difference.vector(xyz, xyz.inds=NULL, normalize=FALSE)
```

Arguments

xyz	numeric matrix of Cartesian coordinates with a row per structure.
xyz.inds	a vector of indices that selects the elements of columns upon which the calculation should be based.
normalize	logical, if TRUE the difference vector is normalized.

Details

Squared overlap (or dot product) is used to measure the similiarity between a displacement vector (e.g. a difference vector between two conformational states) and mode vectors obtained from principal component or normal modes analysis.

Value

Returns a numeric vector of the structural difference (normalized if desired).

Author(s)

Lars Skjaerven

References

Grant, B.J. et al. (2006) *Bioinformatics* **22**, 2695–2696.

See Also

overlap

Examples

```
data(kinesin)
attach(kinesin, warn.conflicts=FALSE)

# Ignore gap containing positions
gaps.pos <- gap.inspect(pdbs$xyz)

#-- Do PCA
pc.xray <- pca.xyz(pdbs$xyz[, gaps.pos$f.inds])

# Define a difference vector between two structural states
diff.inds <- c(grep("d1v8ja", pdbs$id),
               grep("d1goja", pdbs$id))

## Calculate the difference vector
dv <- difference.vector( pdbs$xyz[diff.inds,], gaps.pos$f.inds )

# Calculate the squared overlap between the PCs and the difference vector
o <- overlap(pc.xray, dv)
```

---

dist.xyz *Calculate the Distances Between the Rows of Two Matrices*

Description

Compute the pairwise euclidean distances between the rows of two matrices.

Usage

```
dist.xyz(a, b = NULL, all.pairs=TRUE, ncore=1, nseg.scale=1)
```

Arguments

a	a numeric data matrix of vector
b	an optional second data matrix or vector
all.pairs	logical, if TRUE all pairwise distances between the rows of 'a' and all rows of 'b' are computed, if FALSE only the distances between coresponding rows of 'a' and 'b' are computed.
ncore	number of CPU cores used to do the calculation. ncore>1 requires package 'parallel' installed.

`nseg.scale` split input data into specified number of segments prior to running multiple core calculation. See `fit.xyz`.

Details

This function returns a matrix of euclidean distances between each row of 'a' and all rows of 'b'. Input vectors are coerced to three dimensional matrices (representing the Cartesian coordinates x, y and z) prior to distance computation. If 'b' is not provided then the pairwise distances between all rows of 'a' are computed.

Value

Returns a matrix of pairwise euclidean distances between each row of 'a' and all rows of 'b'.

Note

This function will choke if 'b' has too many rows.

Author(s)

Barry Grant

References

Grant, B.J. et al. (2006) *Bioinformatics* **22**, 2695–2696.

See Also

`dm`, `dist`

Examples

```
dist.xyz( c(1,1,1, 3,3,3), c(3,3,3, 2,2,2, 1,1,1))
dist.xyz( c(1,1,1, 3,3,3), c(3,3,3, 2,2,2, 1,1,1), all.pairs=FALSE)
```

`dm` *Distance Matrix Analysis*

Description

Construct a distance matrix for a given protein structure.

Usage

```
dm(pdb, selection = "calpha", verbose=TRUE)
dm.xyz(xyz, grpby = NULL, scut = NULL, mask.lower = TRUE)
```

Arguments

pdb	a pdb structure object as returned by read.pdb or a numeric vector of 'xyz' coordinates.
selection	a character string for selecting the pdb atoms to undergo comparison (see atom.select).
verbose	logical, if TRUE possible warnings are printed.
xyz	a numeric vector of Cartesian coordinates.
grpby	a vector counting connective duplicated elements that indicate the elements of xyz that should be considered as a group (e.g. atoms from a particular residue).
scut	a cutoff neighbour value which has the effect of excluding atoms, or groups, that are sequentially within this value.
mask.lower	logical, if TRUE the lower matrix elements (i.e. those below the diagonal) are returned as NA.

Details

Distance matrices, also called distance plots or distance maps, are an established means of describing and comparing protein conformations (e.g. Phillips, 1970; Holm, 1993).

A distance matrix is a 2D representation of 3D structure that is independent of the coordinate reference frame and, ignoring chirality, contains enough information to reconstruct the 3D Cartesian coordinates (e.g. Havel, 1983).

Value

Returns a numeric matrix of class "dmat", with all N by N distances, where N is the number of selected atoms.

Note

The input selection can be any character string or pattern interpretable by the function atom.select.
For example, shortcuts "calpha", "back", "all" and selection strings of the form /segment/chain/residue number/resi
name/element number/element name/; see atom.select for details.

If a coordinate vector is provided as input (rather than a pdb object) the selection option is redundant and the input vector should be pruned instead to include only desired positions.

Author(s)

Barry Grant

References

Grant, B.J. et al. (2006) *Bioinformatics* **22**, 2695–2696.

Phillips (1970) *Biochem. Soc. Symp.* **31**, 11–28.

Holm (1993) *J. Mol. Biol.* **233**, 123–138.

Havel (1983) *Bull. Math. Biol.* **45**, 665–720.

See Also

plot.dmat, read.pdb, atom.select

Examples

```
##--- Distance Matrix Plot
pdb <- read.pdb( "4q21" )
k <- dm(pdb,selection="calpha")
filled.contour(k, nlevels = 10)

## NOTE: FOLLOWING EXAMPLE NEEDS MUSCLE INSTALLED
if(check.utility("muscle")) {

##--- DDM: Difference Distance Matrix
# Downlaod and align two PDB files
pdbs <- pdbaln( get.pdb( c( "4q21", "521p"), path = tempdir() ), outfile = tempfile() )

# Get distance matrix
a <- dm(pdbs$xyz[1,])
b <- dm(pdbs$xyz[2,])

# Calculate DDM
c <- a - b

# Plot DDM
plot(c,key=FALSE, grid=FALSE)

plot(c, axis.tick.space=10,
     resnum.1=pdbs$resno[1,],
     resnum.2=pdbs$resno[2,],
     grid.col="black",
     xlab="Residue No. (4q21)", ylab="Residue No. (521p)")
}


## Not run:
##-- Residue-wise distance matrix based on the
##   minimal distance between all available atoms
l <- dm.xyz(pdb$xyz, grpby=pdb$atom[,"resno"], scut=3)

## End(Not run)
```

---

dssp *Secondary Structure Analysis with DSSP or STRIDE*

Description

Secondary structure assignment according to the method of Kabsch and Sander (DSSP) or the method of Frishman and Argos (STRIDE).

Usage

```
dssp(pdb, exefile = "dssp", resno=TRUE, full=FALSE, verbose=FALSE)
stride(pdb, exefile = "stride", resno=TRUE)
## S3 method for class 'sse'
print(x, ...)
```

Arguments

pdb	a structure object of class "pdb", obtained from read.pdb.
exefile	file path to the 'DSSP' or 'STRIDE' program on your system (i.e. how is 'DSSP' or 'STRIDE' invoked).
resno	logical, if TRUE output is in terms of residue numbers rather than residue index (position in sequence).
full	logical, if TRUE bridge pairs and hbonds columns are parsed.
verbose	logical, if TRUE 'DSSP' warning and error messages are printed.
x	an sse object obtained from dssp or stride.
...	additional arguments to 'print'.

Details

This function calls the 'DSSP' or 'STRIDE' program to define secondary structure and psi and phi torsion angles.

Value

Returns a list with the following components:

helix	'start', 'end', 'length', 'chain' and 'type' of helix, where start and end are residue numbers or residue index positions depending on the value of "resno" input argument.
sheet	'start', 'end' and 'length' of E type sse, where start and end are residue numbers "resno".
turn	'start', 'end' and 'length' of T type sse, where start and end are residue numbers "resno".
phi	a numeric vector of phi angles.
psi	a numeric vector of psi angles.
acc	a numeric vector of solvent accessibility.
sse	a character vector of secondary structure type per residue.
hbonds	a 10 or 16 column matrix containing the bridge pair records as well as backbone NH–>O and O–>NH H-bond records. (Only available for dssp

Note

A system call is made to the 'DSSP' or 'STRIDE' program, which must be installed on your system and in the search path for executables.

For the hbonds list component the column names can be used as a convenient means of data access, namely:
Bridge pair 1 "BP1",
Bridge pair 2 "BP2",
Backbone H-bond (NH–>O) "NH-O.1",
H-bond energy of NH–>O "E1",
Backbone H-bond (O–>NH) "O-HN.1",
H-bond energy of O–>NH "E2",
Backbone H-bond (NH–>O) "NH-O.2",
H-bond energy of NH–>O "E3",
Backbone H-bond (O–>NH) "O-HN.2",
H-bond energy of O–>NH "E4".

If 'resno=TRUE' the following additional columns are included:
Chain ID of resno "BP1": "ChainBP1",
Chain ID of resno "BP2": "ChainBP2",
Chain ID of resno "O-HN.1": "Chain1",
Chain ID of resno "NH-O.2": "Chain2",
Chain ID of resno "O-HN.1": "Chain3",
Chain ID of resno "NH-O.2": "Chain4".

Author(s)

Barry Grant

References

Grant, B.J. et al. (2006) *Bioinformatics* **22**, 2695–2696.

'DSSP' is the work of Kabsch and Sander: Kabsch and Sander (1983) *Biopolymers.* **12**, 2577–2637.

For information on obtaining 'DSSP', see:
http://swift.cmbi.ru.nl/gv/dssp/.

'STRIDE' is the work of Frishman and Argos: Frishman and Argos (1995) *Proteins.* **3**, 566–579.

For information on obtaining the 'STRIDE' program, see:
http://webclu.bio.wzw.tum.de/stride/, or copy it from an installation of VMD.

See Also

read.pdb, torsion.pdb, torsion.xyz, plot.bio3d

Examples

```
## Not run:
# Read a PDB file
pdb <- read.pdb("1bg2")
```

```
sse <- dssp(pdb)
sse2 <- stride(pdb)

## Short summary
sse
sse2

# Helix data
sse$helix

# Precent SSE content
sum(sse$helix$length)/sum(pdb$calpha) * 100
sum(sse$sheet$length)/sum(pdb$calpha) * 100


## End(Not run)
```

`dssp.pdbs` *Secondary Structure Analysis of Aligned PDB Structures with DSSP*

Description

Secondary structure assignment of aligned PDB structures - according to the method of Kabsch and Sander (DSSP)

Usage

```
dssp.pdbs(pdbs)
```

Arguments

`pdbs`	a list object of class `"pdbs"` (obtained with `pdbaln` or `read.fasta.pdb`).

Details

This is a wrapper function for easy SSE assignment of aligned PDB structures.

Value

Returns a character matrix of aligned SSE assignment corresponding to each structure in the pdbs object (row wise).

Author(s)

Lars Skjaerven

References

Grant, B.J. et al. (2006) *Bioinformatics* **22**, 2695–2696.

See Also

dssp, pdbaln

Examples

```
## Not run:
## Fetch PDB files and split to chain A only PDB files
ids <- c("1a70_A", "1czp_A", "1frd_A", "1fxi_A", "1iue_A", "1pfd_A")
raw.files <- get.pdb(ids, path = "raw_pdbs")
files <- pdbsplit(raw.files, ids, path = "raw_pdbs/split_chain")

## Sequence Alignement
pdbs <- pdbaln(files)

## SSE assignment
sse <- dssp.pdbs(pdbs)

## End(Not run)
```

---

dssp.trj *Secondary Structure Analysis of Trajectories with DSSP*

Description

Secondary structure assignment according to the method of Kabsch and Sander.

Usage

```
dssp.trj(pdb, trj, skip=1000, threshold=3, file.head="")
```

Arguments

pdb	a structure object of class "pdb", obtained from read.pdb.
trj	a trajectory object of class "trj", obtained from read.ncdf, read.dcd, read.crd.
skip	a number indicating the frame frequency in the trajectory indicated; default = 1000.
threshold	a number indicating the threshold to be used for the analysis: higher numbers will decrease the threshold, allowing to accept structures with smaller secondary structure differences, with respect of the reference pdb structure; default = 3
file.head	a path where to save the structures with possible unfolding events.

Details

This function calls the 'DSSP' program and calculates the secondary structure difference between the reference pdb and frames from the trj file indicated, according with the skip value indicated.

Value

This function gives, as output, a comparative secondary structure analysis between different structures: particularly, the frame number under analysis will be printed on screen if the structure shows possible unfolding events and the frame structure will be saved, according with the `file.head` indicated.

Note

A system call is made to the 'DSSP' program, which must be installed on your system and in the search path for executables.

Author(s)

Barry Grant

References

Grant, B.J. et al. (2006) *Bioinformatics* **22**, 2695–2696.

'DSSP' is the work of Kabsch and Sander: Kabsch and Sander (1983) *Biopolymers.* **12**, 2577–2637.

For information on obtaining 'DSSP', see:
http://swift.cmbi.ru.nl/gv/dssp/.

See Also

`read.pdb`, `read.ncdf`, `read.dcd`, `read.crd`, `plot.bio3d`, `dssp`

Examples

```
## Not run:
# Read a PDB file
pdb <- read.pdb(system.file("examples/hivp.pdb", package="bio3d"))
trj <- read.dcd(system.file("examples/hivp.dcd", package="bio3d"))

dssp.trj(pdb, trj, skip = 100, threshold = 1.5)


## End(Not run)
```

---

elements *Periodic Table of the Elements*

Description

This data set gives various information on chemical elements.

Usage

```
elements
```

Format

A data frame containing for each chemical element the following information.

`num` atomic number
`symb` elemental symbol
`areneg` Allred and Rochow electronegativity (0.0 if unknown)
`rcov` covalent radii (in Angstrom) (1.6 if unknown)
`rbo` "bond order" radii
`rvdw` van der Waals radii (in Angstrom) (2.0 if unknown)
`maxbnd` maximum bond valence (6 if unknown)
`mass` IUPAC recommended atomic masses (in amu)
`elneg` Pauling electronegativity (0.0 if unknown)
`ionization` ionization potential (in eV) (0.0 if unknown)
`elaffinity` electron affinity (in eV) (0.0 if unknown)
`red` red value for visualization
`green` green value for visualization
`blue` blue value for visualization
`name` element name

Source

Open Babel (2.3.1) file: element.txt

Created from the Blue Obelisk Cheminformatics Data Repository
Direct Source: http://www.blueobelisk.org/
http://www.blueobelisk.org/repos/blueobelisk/elements.xml includes furhter bibliographic citation information

- Allred and Rochow Electronegativity from http://www.hull.ac.uk/chemistry/electroneg.php?type=Allred-Rochow
- Covalent radii from http://dx.doi.org/10.1039/b801115j
- Van der Waals radii from http://dx.doi.org/10.1021/jp8111556

Examples

```
data(elements)
elements

# Get the mass of some elements
symb <- c("C","O","H")
elements[match(symb,elements[,"symb"]),"mass"]

# Get the van der Waals radii of some elements
symb <- c("C","O","H")
elements[match(symb,elements[,"symb"]),"rvdw"]
```

entropy *Shannon Entropy Score*

Description

Calculate the sequence entropy score for every position in an alignment.

Usage

```
entropy(alignment)
```

Arguments

alignment	sequence alignment returned from read.fasta or an alignment character matrix.

Details

Shannon's information theoretic entropy (Shannon, 1948) is an often-used measure of residue diversity and hence residue conservation.

Value

Returns a list with five components:

H	standard entropy score for a 22-letter alphabet.
H.10	entropy score for a 10-letter alphabet (see below).
H.norm	normalized entropy score (for 22-letter alphabet), so that conserved (low entropy) columns (or positions) score 1, and diverse (high entropy) columns score 0.
H.10.norm	normalized entropy score (for 10-letter alphabet), so that conserved (low entropy) columns score 1 and diverse (high entropy) columns score 0.
freq	residue frequency matrix containing percent occurrence values for each residue type.

Note

In addition to the standard entropy score (based on a 22-letter alphabet of the 20 standard aminoacids, plus a gap character '-' and a mask character 'X'), an entropy score, H.10, based on a 10-letter alphabet is also returned.

For H.10, residues from the 22-letter alphabet are classified into one of 10 types, loosely following the convention of Mirny and Shakhnovich (1999): Hydrophobic/Aliphatic [V,I,L,M], Aromatic [F,W,Y], Ser/Thr [S,T], Polar [N,Q], Positive [H,K,R], Negative [D,E], Tiny [A,G], Proline [P], Cysteine [C], and Gaps [-,X].

The residue code 'X' is useful for handling non-standard aminoacids.

Author(s)

Barry Grant

References

Grant, B.J. et al. (2006) *Bioinformatics* **22**, 2695–2696.

Shannon (1948) *The System Technical J.* **27**, 379–422.

Mirny and Shakhnovich (1999) *J. Mol. Biol.* **291**, 177–196.

See Also

consensus, read.fasta

Examples

```
# Read HIV protease alignment
aln <- read.fasta(system.file("examples/hivp_xray.fa",package="bio3d"))

# Entropy and consensus
h   <- entropy(aln)
con <- consensus(aln)

names(h$H)=con$seq
print(h$H)

# Entropy for sub-alignment (positions 1 to 20)
h.sub <- entropy(aln$ali[,1:20])

# Plot entropy and residue frequencies (excluding positions >=60 percent gaps)
H <- h$H.norm
H[ apply(h$freq[21:22,],2,sum)>=0.6 ] = 0

col <- mono.colors(32)
aa  <- rev(rownames(h$freq))
oldpar <- par(no.readonly=TRUE)
layout(matrix(c(1,2),2,1,byrow = TRUE), widths = 7,
       heights = c(2, 8), respect = FALSE)

# Plot 1: entropy
par(mar = c(0, 4, 2, 2))
barplot(H, border="white", ylab = "Entropy",
        space=0, xlim=c(3.7, 97.3),yaxt="n" )
axis(side=2, at=c(0.2,0.4, 0.6, 0.8))
axis(side=3, at=(seq(0,length(con$seq),by=5)-0.5),
     labels=seq(0,length(con$seq),by=5))
box()

# Plot2: residue frequencies
par(mar = c(5, 4, 0, 2))
image(x=1:ncol(con$freq),
```

```
      y=1:nrow(con$freq),
      z=as.matrix(rev(as.data.frame(t(con$freq)))),
      col=col, yaxt="n", xaxt="n",
      xlab="Alignment Position", ylab="Residue Type")
axis(side=1, at=seq(0,length(con$seq),by=5))
axis(side=2, at=c(1:22), labels=aa)
axis(side=3, at=c(1:length(con$seq)), labels =con$seq)
axis(side=4, at=c(1:22), labels=aa)
grid(length(con$seq), length(aa))
box()

for(i in 1:length(con$seq)) {
  text(i, which(aa==con$seq[i]),con$seq[i],col="white")
}
abline(h=c(3.5, 4.5, 5.5, 3.5, 7.5, 9.5,
         12.5, 14.5, 16.5, 19.5), col="gray")

par(oldpar)
```

`example.data` *Bio3d Example Data*

Description

These data sets contain the results of running various Bio3D functions on example kinesin and transducin structural data, and on a short coarse-grained MD simulation data for HIV protease. The main purpose of including this data (which may be generated by the user by following the extended examples documented within the various Bio3D functions) is to speed up example execution. It should allow users to more quickly appreciate the capabilities of functions that would otherwise require raw data download, input and processing before execution.

Note that related datasets formed the basis of the work described in (Grant, 2007) and (Yao & Grant, 2013) for `kinesin` and `transducin` examples, respectively.

Usage

```
data(kinesin)
data(transducin)
data(hivp)
```

Format

Three objects from analysis of the `kinesin` and `transducin` sequence and structure data:

1. pdbs is a list of class `"pdbs"` containing aligned PDB structure data. In the case of transducin this is the output of running `pdbaln` on a set of 47 kinesin structures from the SCOP database (again see `pdbs$id` for details). In both cases the coordinates are fitted onto the first structure based on `"core"` positions obtained from `core.find` and superposed using the function `pdbfit`.

2. core is a list of class "core" obtained by running the function core.find on the pdbs object as described above.
3. annotation is a character matrix describing the nucleotide state and bound ligand species for each structure in pdbs as obtained from the function pdb.annotate.

One object named net in the hivp example data stores the correlation network obtained from the analysis of the MD simulation trajectory of HIV protease using the cna function. The original trajectory file can be accessed by the command 'system.file("examples/hivp.dcd", package="bio3d")'.

Source

A related but more extensive dataset formed the basis of the work described in (Grant, 2007) and (Yao & Grant, 2013) for kinesin and transducin examples, respectively.

References

Grant, B.J. et al. (2006) *Bioinformatics* **22**, 2695–2696.

Grant, B.J. et al. (2007) *J. Mol. Biol.* **368**, 1231–1248.

Yao, X.Q. et al. (2013) *Biophys. J.* **105**, L08–L10.

---

filter.dccm *Filter for Cross-correlation Matrices (Cij)*

Description

This function builds various cij matrix for correlation network analysis

Usage

```
filter.dccm(x, cutoff.cij = 0.4, cmap = NULL, xyz = NULL, fac = NULL,
    cutoff.sims = NULL, collapse = TRUE, extra.filter = NULL, ...)
```

Arguments

x	A matrix (nXn), a numeric array with 3 dimensions (nXnXm), a list with m cells each containing nXn matrix, or a list with 'all.dccm' component, containing atomic correlation values, where "n" is the number of residues and "m" the number of calculations. The matrix elements should be in between -1 and 1. See 'dccm' function in bio3d package for further details.
cutoff.cij	Threshold for each individual correlation value. See below for details.
cmap	logical, if TRUE both correlation values and contact map are inspected.
xyz	XYZ coordinates for distance matrix calculation.
fac	factor indicating distinct categories of input correlation matrices.
cutoff.sims	Threshold for the number of simulations with observed correlation value above cutoff.cij for the same residue/atomic pairs. See below for details.

collapse	logical, if TRUE the mean matrix will be returned.
extra.filter	Filter to apply in addition to the model chosen.
...	extra arguments passed to function cmap.

Details

If cmap=TRUE, the function inspects a set of cross-correlation matrices, or DCCM, and decides edges for correlation network analysis based on: 1. min(abs(cij)) >= cutoff.cij, or 2. max(abs(cij)) >= cutoff.cij && residues contact each other based on results from cmap.

If cmap=FALSE, the function filters DCCMs with cutoff.cij and return the mean of correlations present in at least cutoff.sims calculated matrices.

Value

Returns a matrix of class "dccm" or a 3D array of filtered cross-correlations.

Author(s)

Xin-Qiu Yao, Guido Scarabelli & Barry Grant

References

Grant, B.J. et al. (2006) *Bioinformatics* **22**, 2695–2696.

See Also

cna, dccm, dccm.nma, dccm.xyz, cmap, plot.dccm

Examples

```
## Not run:

# Example of transducin
data(transducin)
attach(transducin)

gaps.pos <- gap.inspect(pdbs$xyz)
modes <- nma.pdbs(pdbs, full=TRUE)
dccms <- dccm.enma(modes)

cij <- filter.dccm(dccms, xyz=pdbs)

# Example protein kinase
# Select Protein Kinase PDB IDs
ids <- c("4b7t_A", "2exm_A", "1opj_A", "4jaj_A", "1a9u_A",
                "1tki_A", "1csn_A", "1lp4_A")

# Download and split by chain ID
files <- get.pdb(ids, path = "raw_pdbs", split=TRUE)
```

```
# Alignment of structures
pdbs <- pdbaln(files) # Sequence identity
summary(c(seqidentity(pdbs)))

# NMA on all structures
modes <- nma.pdbs(pdbs, full = TRUE)

# Calculate correlation matrices for each structure
cij <- dccm(modes)

# Set DCCM plot panel names for combined figure
dimnames(cij$all.dccm) = list(NULL, NULL, ids)
plot.dccm(cij$all.dccm)

# Filter to display only correlations present in all structures
cij.all <- filter.dccm(cij, cutoff.sims = 8, cutoff.cij = 0)
plot.dccm(cij.all, main = "Consensus Residue Cross Correlation")


## End(Not run)
```

`fit.xyz` *Coordinate Superposition*

Description

Coordinate superposition with the Kabsch algorithm.

Usage

```
fit.xyz(fixed, mobile,
        fixed.inds  = NULL,
        mobile.inds = NULL,
        verbose=FALSE,
        prefix= "", pdbext = "",
        outpath = "fitlsq", full.pdbs=FALSE,
        ncore = 1, nseg.scale = 1, ...)

rot.lsq(xx, yy,
        xfit = rep(TRUE, length(xx)), yfit = xfit,
        verbose = FALSE)
```

Arguments

`fixed`	numeric vector of xyz coordinates.
`mobile`	numeric vector, numeric matrix, or an object with an `xyz` component containing one or more coordinate sets.

fixed.inds	a vector of indices that selects the elements of fixed upon which fitting should be based.
mobile.inds	a vector of indices that selects the elements of mobile upon which fitting should be based.
full.pdbs	logical, if TRUE "full" coordinate files (i.e. all atoms) are written to the location specified by outpath.
prefix	prefix to mobile$id to locate "full" input PDB files. Only required if full.pdbs is TRUE.
pdbext	the file name extension of the input PDB files.
outpath	character string specifing the output directory when full.pdbs is TRUE.
xx	numeric vector corresponding to the moving 'subject' coordinate set.
yy	numeric vector corresponding to the fixed 'target' coordinate set.
xfit	logical vector with the same length as xx, with TRUE elements corresponding to the subset of positions upon which fitting is to be performed.
yfit	logical vector with the same length as yy, with TRUE elements corresponding to the subset of positions upon which fitting is to be performed.
verbose	logical, if TRUE more details are printed.
...	other parameters for read.pdb.
ncore	number of CPU cores used to do the calculation. ncore>1 requires package 'parallel' installed.
nseg.scale	split input data into specified number of segments prior to running multiple core calculation.

Details

The function fit.xyz is a wrapper for the function rot.lsq, which performs the actual coordinate superposition. The function rot.lsq is an implementation of the Kabsch algorithm (Kabsch, 1978) and evaluates the optimal rotation matrix to minimize the RMSD between two structures.

Since the Kabsch algorithm assumes that the number of points are the same in the two input structures, care should be taken to ensure that consistent atom sets are selected with fixed.inds and mobile.inds.

Optionally, "full" PDB file superposition and output can be accomplished by setting full.pdbs=TRUE. In that case, the input (mobile) passed to fit.xyz should be a list object obtained with the function read.fasta.pdb, since the components id, resno and xyz are required to establish correspondences. See the examples below.

In dealing with large vector and matrix, running on multiple cores, especially when ncore>>1, may ask for a large portion of system memory. To avoid the overuse of memory, input data is first split into segments (for xyz matrix, the splitting is along the row). The number of data segments is equal to nseg.scale*nseg.base, where nseg.base is an integer determined by the dimension of the data.

Value

Returns moved coordinates.

Author(s)

Barry Grant with `rot.lsq` contributions from Leo Caves

References

Grant, B.J. et al. (2006) *Bioinformatics* **22**, 2695–2696.

Kabsch *Acta Cryst* (1978) **A34**, 827–828.

See Also

`rmsd`, `read.pdb`, `read.fasta.pdb`, `read.dcd`

Examples

```
##--- Read an alignment & Fit aligned structures
aln  <- read.fasta(system.file("examples/kif1a.fa",package="bio3d"))
pdbs <- read.fasta.pdb(aln)

gaps <- gap.inspect(pdbs$xyz)
rmsd( pdbs$xyz[, gaps$f.inds] )

xyz <- fit.xyz( fixed  = pdbs$xyz[1,],
                mobile = pdbs$xyz,
                fixed.inds  = gaps$f.inds,
                mobile.inds = gaps$f.inds )

rmsd( xyz[, gaps$f.inds] )



##-- Superpose again this time outputing PDBs
## Not run:
xyz <- fit.xyz( fixed = pdbs$xyz[1,],
                mobile = pdbs,
                fixed.inds  = gaps$f.inds,
                mobile.inds = gaps$f.inds,
                outpath = "rough_fit",
                full.pdbs = TRUE)

##--- Fit two PDBs
A <- read.pdb("1bg2")
A.ind <- atom.select(A, "///256:269///CA/")

B <- read.pdb("2kin")
B.ind <- atom.select(B, "///257:270///CA/")

xyz <- fit.xyz(fixed=A$xyz, mobile=B$xyz,
                fixed.inds=A.ind$xyz,
                mobile.inds=B.ind$xyz)
```

```
# Write out moved PDB
C <- B; C$xyz = xyz
write.pdb(pdb=C, file = "moved.pdb")

## End(Not run)
```

`fluct.nma` *NMA Fluctuations*

Description

Calculates the atomic fluctuations from normal modes analysis.

Usage

```
fluct.nma(nma, mode.inds=NULL)
```

Arguments

`nma`	a list object of class `"nma"` (obtained with `nma`).
`mode.inds`	a numeric vector containing the the mode numbers in which the calculation should be based.

Details

Atomic fluctuations are calculated based on the `nma` object. By default all modes are included in the calculation.

See examples for more details.

Value

Returns a numeric vector of atomic fluctuations.

Author(s)

Lars Skjaerven

References

Hinsen, K. et al. (2000) *Chemical Physics* **261**, 25–37. Grant, B.J. et al. (2006) *Bioinformatics* **22**, 2695–2696.

See Also

`nma`

Examples

```
## Fetch stucture
pdb <- read.pdb( system.file("examples/1hel.pdb", package="bio3d") )

## Calculate (vibrational) normal modes
modes <- nma(pdb)

## Fluctuations
f <- fluct.nma(modes)

## Fluctuations of first non-trivial mode
f <- fluct.nma(modes, mode.inds=c(7,8))
```

`formula2mass` *Chemical Formula to Mass Converter*

Description

Compute the molar mass associated to a chemical formula.

Usage

```
formula2mass(form, sum.mass = TRUE)
```

Arguments

`form`	a character string containing a chemical formula on the form: 'C3 H5 N O1'.
`sum.mass`	logical, should the mass of each element be summed.

Details

Compute the molar mass (in g.mol-1) associated to a chemical formula.

Value

Return a single element numeric vector containing the mass corresponding to a given chemical formula.

Author(s)

Lars Skjaerven

See Also

`atom2ele`, `atom2mass`

Examples

```
#formula2mass("C5 H6 N O3")
```

`gap.inspect` *Alignment Gap Summary*

Description

Report the number of gaps per sequence and per position for a given alignment.

Usage

```
gap.inspect(x)
```

Arguments

`x`	a matrix or an alignment data structure obtained from `read.fasta` or `read.fasta.pdb`.

Details

Reports the number of gap characters per row (i.e. sequence) and per column (i.e. position) for a given `alignment`. In addition, the indices for gap and non-gap containing coloums are returned along with a binary matrix indicating the location of gap positions.

Value

Returns a list object with the following components:

`row`	a numeric vector detailing the number of gaps per row (i.e. sequence).
`col`	a numeric vector detailing the number of gaps per column (i.e. position).
`t.inds`	indices for gap containing coloums
`f.inds`	indices for non-gap containing coloums
`bin`	a binary numeric matrix with the same dimensions as the `alignment`, with 0 at non-gap positions and 1 at gap positions.

Note

During alignment, gaps are introduced into sequences that are believed to have undergone deletions or insertions with respect to other sequences in the alignment. These gaps, often referred to as indels, can be represented with 'NA', a '-' or '.' character.

This function gives an overview of gap occurrence and may be useful when considering positions or sequences that could/should be excluded from further analysis.

Author(s)

Barry Grant

References

Grant, B.J. et al. (2006) *Bioinformatics* **22**, 2695–2696.

See Also

read.fasta, read.fasta.pdb

Examples

```
aln <- read.fasta( system.file("examples/hivp_xray.fa",
                   package = "bio3d") )

gap.stats <- gap.inspect(aln$ali)
gap.stats$row # Gaps per sequence
gap.stats$col # Gaps per position
##gap.stats$bin # Binary matrix (1 for gap, 0 for aminoacid)
##aln[,gap.stats$f.inds] # Alignment without gap positions

plot(gap.stats$col, typ="h", ylab="No. of Gaps")
```

---

geostas *GeoStaS Domain Finder*

Description

Identifies geometrically stable domains in biomolecules

Usage

```
geostas(xyz, amsm = NULL, k = 3, method = "pairwise", fit = TRUE, ...)

amsm.xyz(xyz, ncore = NULL)
```

Arguments

xyz	numeric matrix of xyz coordinates as obtained e.g. by read.ncdf, read.dcd, or mktrj.
amsm	a numeric matrix as obtained by amsm.xyz (convenient e.g. for re-doing only the clustering analysis of the 'AMSM' matrix).
k	an integer scalar or vector with the desired number of groups.
method	character string specifing the approach for clustering the atomic movement similarity matrix (pariwise or columnwise).
fit	logical, if TRUE coordinate superposition on identified core atoms is performed prior to the calculation of the AMS matrix.
ncore	number of CPU cores used to do the calculation. ncore>1 requires package 'parallel' installed.
...	additional arguments to amsm.xyz.

Details

This function attempts to identify rigid domains in a protein (or nucleic acid) structure based on an structural ensemble, e.g. obtained from NMR experiments, molecular dynamics simulations, or normal mode analysis.

The algorithm is based on a geometric approach for comparing pairwise traces of atomic motion and the search for their best superposition using a quaternion representation of rotation. The result is stored in a NxN atomic movement similarity matrix (AMSM) describing the correspondence between all pairs of atom motion. Rigid domains are obtained by clustering the elements of the AMS matrix (`method=pairwise`), or alternatively, the columns similarity (`method=columnwise`), using K-means clustering (see function 'kmeans()').

Compared to the conventional cross-correlation matrix (see function `dccm`) the "geostas" approach provide functionality to also detect domains involved in rotational motions (i.e. two atoms located on opposite sides of a rotating domain will appear as anti-correlated in the cross-correlation matrix, but should obtain a high similarity coefficient in the AMS matrix).

See examples for more details.

Value

Returns a list object with the following components:

`grps`	a numeric vector containing the domain assignment per residue.
`amsm`	a numeric matrix of atomic movement similarity (AMSM).
`fit.inds`	a numeric vector of xyz indices used for fitting.

Note

The current implementation in Bio3D uses a different fitting and clustering approach than the original Java implementation. The results will therefore differ.

Author(s)

Julia Romanowska and Lars Skjaerven

References

Romanowska, J. et al. (2012) *JCTC* **8**, 2588–2599. Grant, B.J. et al. (2006) *Bioinformatics* **22**, 2695–2696.

See Also

`read.ncdf`, `read.dcd`, `read.pdb`, `mktrj`.

Examples

```
#### NMR-ensemble example
## Read a multi-model PDB file
```

```
pdb <- read.pdb("1d1d", multi=TRUE)
ca.inds <- atom.select(pdb, 'calpha')

## Find domains and write PDB
gs  <- geostas(pdb)

## Plot a atomic movement similarity matrix
plot.dccm(gs$amsm, at=seq(0, 1, 0.1), main="AMSM with Domain Assignment",
          col.regions=rev(heat.colors(200)), margin.segments=gs$grps, contour=FALSE)

## Fit all frames to the 'first' domain
domain.inds <- atom2xyz(which(gs$grps==1))

xyz <- fit.xyz(pdb$xyz[1, ca.inds$xyz],
               pdb$xyz[, ca.inds$xyz],
               fixed.inds = domain.inds,
               mobile.inds = domain.inds)

#write.pdb(xyz=xyz, chain=gs$grps)



## Not run:
#### NMA example
## Fetch stucture
pdb <- read.pdb("1crn")

## Calculate (vibrational) normal modes
modes <- nma(pdb)

## Mode trajectories
trj <- rbind(mktrj(modes, mode=7),
             mktrj(modes, mode=8),
             mktrj(modes, mode=9),
             mktrj(modes, mode=10),
             mktrj(modes, mode=11))

## Find domains
gs <- geostas(trj, k=2)

## Write NMA trajectory with domain assignment
mktrj(modes, mode=7, chain=gs$grps)

## Redo geostas domain clustering
gs <- geostas(trj, amsm=gs$amsm, k=5)




#### Trajectory example
## Read inn DCD trajectory file, fit coordinates
dcdfile <- system.file("examples/hivp.dcd", package = "bio3d")
trj <- read.dcd(dcdfile)
```

```
xyz <- fit.xyz(trj[1,], trj)

## Find domains
gs <- geostas(xyz, k=3, fit=FALSE)

## Principal component analysis
pc.md <- pca.xyz(xyz)

## Visualize PCs with colored domains (chain ID)
mktrj(pc.md, pc=1, chain=gs$grps)




#### X-ray ensemble GroEL subunits
# Define the ensemble PDB-ids
ids <- c("1sx4_[A,B,H,I]", "1xck_[A-B]", "1sx3_[A-B]", "4ab3_[A-B]")

# Download and split PDBs by chain ID
raw.files <- get.pdb(ids, path = "raw_pdbs", gzip = TRUE)
files <- pdbsplit(raw.files, ids, path = "raw_pdbs/split_chain/")

# Align and superimpose coordinates
pdbs <- pdbaln(files)
gs <- geostas(pdbs, k=4, fit=TRUE)

# Principal component analysis
gaps.pos <- gap.inspect(pdbs$xyz)
xyz <- fit.xyz(pdbs$xyz[1, gaps.pos$f.inds],
               pdbs$xyz[, gaps.pos$f.inds],
               fixed.inds=gs$fit.inds,
               mobile.inds=gs$fit.inds)
pc.xray <- pca.xyz(xyz)

## Visualize PCs with colored domains (chain ID)
mktrj(pc.xray, pc=1, chain=gs$grps)


## End(Not run)
```

`get.pdb` *Download PDB Coordinate Files*

Description

Downloads PDB coordinate files from the RCSB Protein Data Bank.

Usage

```
get.pdb(ids, path = ".", URLonly=FALSE, overwrite = FALSE, gzip = FALSE,
     split = FALSE, verbose = TRUE, ncore = 1, ...)
```

Arguments

ids	A character vector of one or more 4-letter PDB codes/identifiers or 6-letter PDB-ID_Chain-ID of the files to be downloaded, or a 'blast' object containing 'pdb.id'.
path	The destination path/directory where files are to be written.
URLonly	logical, if TRUE a character vector containing the URL path to the online file is returned and files are not downloaded. If FALSE the files are downloaded.
overwrite	logical, if FALSE the file will not be downloaded if it alread exist.
gzip	logical, if TRUE the gzipped PDB will be downloaded and extracted locally.
split	logical, if TRUE pdbsplit funciton will be called to split pdb files into separated chains.
verbose	print details of the reading process.
ncore	number of CPU cores used to do the calculation. ncore>1 requires package 'parallel' installed.
...	extra arguments passed to pdbsplit function.

Details

This is a basic function to automate file download from the PDB.

Value

Returns a list of successfully downloaded files. Or optionally if URLonly is TRUE a list of URLs for said files.

Author(s)

Barry Grant

References

Grant, B.J. et al. (2006) *Bioinformatics* **22**, 2695–2696.

For a description of PDB format (version3.3) see:
http://www.wwpdb.org/documentation/format33/v3.3.html.

See Also

read.pdb, write.pdb, atom.select, read.fasta.pdb, read.fasta, pdbsplit

Examples

```
## PDB file paths
get.pdb( c("1poo", "1moo"), URLonly=TRUE )

## These URLs can be used by 'read.pdb'
```

```
pdb <- read.pdb( get.pdb("5p21", URL=TRUE) )
summary(pdb)

## Download PDB file
## get.pdb("5p21")
```

`get.seq` *Download FASTA Sequence Files*

Description

Downloads FASTA sequence files from the NR, or SWISSPROT/UNIPROT databases.

Usage

```
get.seq(ids, outfile = "seqs.fasta", db = "nr")
```

Arguments

`ids`	A character vector of one or more appropriate database codes/identifiers of the files to be downloaded.
`outfile`	A single element character vector specifying the name of the local file to which sequences will be written.
`db`	A single element character vector specifying the database from which sequences are to be obtained.

Details

This is a basic function to automate sequence file download from the NR and SWISSPROT/UNIPROT databases.

Value

If all files are successfully downloaded a list object with two components is returned:

`ali`	an alignment character matrix with a row per sequence and a column per equivalent aminoacid/nucleotide.
`ids`	sequence names as identifiers.

This is similar to that returned by `read.fasta`. However, if some files were not successfully downloaded then a vector detailing which ids were not found is returned.

Note

For a description of FASTA format see: http://www.ncbi.nlm.nih.gov/BLAST/blastcgihelp.shtml. When reading alignment files, the dash '-' is interpreted as the gap character.

Author(s)

Barry Grant

References

Grant, B.J. et al. (2006) *Bioinformatics* **22**, 2695–2696.

See Also

blast.pdb, read.fasta, read.fasta.pdb, get.pdb

Examples

```
## Not run:
pdb <- read.pdb( get.pdb("5p21", URLonly=TRUE) )
blast <- blast.pdb( pdbseq(pdb), database = "swissprot" )
ids <- plot.blast( blast )
seq <- get.seq(ids$gi.id, outfile=tempfile())
seq$id[1:10]
seq$ali[1:10,]

## End(Not run)
```

---

hclustplot *Dendrogram with Clustering Annotation*

Description

Draw a standard dendrogram with clustering annotation in the marginal regions and colored labels.

Usage

```
hclustplot(hc, k = NULL, h = NULL, colors = NULL, labels = NULL,
             fillbox = FALSE, heights = c(1, .3), mar = c(1, 1, 0, 1), ...)
```

Arguments

hc	an object of the type produced by hclust.
k	an integer scalar or vector with the desired number of groups. Redirected to function cutree.
h	numeric scalar or vector with heights where the tree should be cut. Redirected to function cutree. At least one of 'k' or 'h' must be specified.
colors	a numerical or character vector with the same length as 'hc' specifying the colors of the labels.
labels	a character vector with the same length as 'hc' containing the labels to be written.

fillbox	logical, if TRUE clustering annotation will be drawn as filled boxes below the dendrogram.
heights	numeric vector of length two specifying the values for the heights of rows on the device. See function layout.
mar	a numerical vector of the form 'c(bottom, left, top, right)' which gives the number of lines of margin to be specified on the four sides of the plot. If left at default the margins will be adjusted upon adding arguments 'main', 'ylab', etc.
...	other graphical parameters passed to functions plot.dendrogram, mtext, and par. Note that certain arguments will be ignored.

Details

This function adds extended visualization of cluster membership to a standard dendrogram. If 'k' or 'h' is provided a call to cutree will provide cluster membership information. Alternatively a vector of colors or cluster membership information can be provided through argument 'colors'.

See examples for further details on usage.

Value

Called for its effect.

Note

Argument 'horiz=TRUE' currently not supported.

Author(s)

Lars Skjaerven

References

Grant, B.J. et al. (2006) *Bioinformatics* **22**, 2695–2696.

See Also

plot.hclust, plot.dendrogram, hclust, cutree.

Examples

```
attach(transducin)

##- perform clustering
rd <- rmsd(pdbs, fit=TRUE)
hc <- hclust(as.dist(rd))

##- draw dendrogram
hclustplot(hc, k=3)
```

```
##- draw dendrogram with manual clustering annotation
hclustplot(hc, colors=annotation[, "color"], labels=pdbs$id)
```

---

hmmer *HMMER Sequence Search*

Description

Run HMMER, against the PDB, NR, swissprot (and other) sequence databases.

Usage

```
hmmer(seq, type="phmmer", db = NULL, verbose = TRUE, timeout = 90)
```

Arguments

seq	a single element or multi-element character vector containing the query sequence. Alternatively a 'fasta' object from function get.seq can be provided.
type	for now: 'phmmer', or 'hmmscan'. Currently not available: 'hmmsearch', 'jackhmmer'
db	a single element character vector specifying the database against which to search. Current options are 'pdb', 'nr' and 'swissprot'. More here...
verbose	logical, if TRUE details of the download process is printed.
timeout	integer specifying the number of seconds to wait for the blast reply before a time out occurs.

Details

This function employs direct HTTP-encoded requests to the HMMER web server ...

Coppy from the HMMER website: HMMER is used for searching sequence databases for homologs of protein sequences, and for making protein sequence alignments. It implements methods using probabilistic models called profile hidden Markov models (profile HMMs).

Value

A data frame with the following components: this depends on the 'type'.

Note

Online access is required to query HMMER services.

Author(s)

Lars Skjaerven

References

Grant, B.J. et al. (2006) *Bioinformatics* **22**, 2695–2696.

Finn, R.D. et al. (2011) *Nucl. Acids Res.* **39**, 29–37. Eddy, S.R. (2011) *PLoS Comput Biol* **7**(10): e1002195.

See also the 'HMMER' website:
http://hmmer.janelia.org

See Also

seqaln

Examples

```
##- PHMMER
seq <- get.seq("2abl_A", outfile=tempfile())
res <- hmmer(seq, db="pdb")

##- HMMSCAN
fam <- hmmer(seq, type="hmmscan", db="pfam")
pfam.aln <- pfam(fam$acc[1])

##- HMMSEARCH
hmm <- hmmer(pfam.aln, type="hmmsearch", db="pdb")
unique(hmm$species)
hmm$acc
```

---

ide.filter *Percent Identity Filter*

Description

Identify and filter subsets of sequences at a given sequence identity cutoff.

Usage

```
ide.filter(aln = NULL, ide = NULL, cutoff = 0.6, verbose = TRUE, ncore=1, nseg.scale=1)
```

Arguments

aln	sequence alignment list, obtained from seqaln or read.fasta, or an alignment character matrix. Not used if 'ide' is given.
ide	an optional identity matrix obtained from seqidentity.
cutoff	a numeric identity cutoff value ranging between 0 and 1.

verbose	logical, if TRUE print details of the clustering process.
ncore	number of CPU cores used to do the calculation. ncore>1 requires package 'parallel' installed.
nseg.scale	split input data into specified number of segments prior to running multiple core calculation. See fit.xyz.

Details

This function performs hierarchical cluster analysis of a given sequence identity matrix 'ide', or the identity matrix calculated from a given alignment 'aln', to identify sequences that fall below a given identity cutoff value 'cutoff'.

Value

Returns a list object with components:

ind	indices of the sequences below the cutoff value.
tree	an object of class "hclust", which describes the tree produced by the clustering process.
ide	a numeric matrix with all pairwise identity values.

Author(s)

Barry Grant

References

Grant, B.J. et al. (2006) *Bioinformatics* **22**, 2695–2696.

See Also

read.fasta, seqaln, seqidentity, entropy, consensus

Examples

```
data(kinesin)
attach(kinesin, warn.conflicts=FALSE)

ide.mat <- seqidentity(pdbs)

# Histogram of pairwise identity values
op <- par(no.readonly=TRUE)
par(mfrow=c(2,1))
hist(ide.mat[upper.tri(ide.mat)], breaks=30,xlim=c(0,1),
     main="Sequence Identity", xlab="Identity")

k <- ide.filter(ide=ide.mat, cutoff=0.6)
ide.cut <- seqidentity(pdbs$ali[k$ind,])
hist(ide.cut[upper.tri(ide.cut)], breaks=10, xlim=c(0,1),
     main="Sequence Identity", xlab="Identity")
```

```
#plot(k$tree, axes = FALSE, ylab="Sequence Identity")
#print(k$ind) # selected
par(op)
detach(kinesin)
```

`identify.cna` *Identify Points in a CNA Protein Structure Network Plot*

Description

'identify.cna' reads the position of the graphics pointer when the (first) mouse button is pressed. It then searches the coordinates given in 'x' for the point closest to the pointer. If this point is close enough to the pointer, its index and community members will be returned as part of the value of the call and the community members will be added as labels to the plot.

Usage

```
  ## S3 method for class 'cna'
identify(x, labels=NULL, cna=NULL, ...)
```

Arguments

`x`	A numeric matrix with Nx2 dimensions, where N is equal to the number of objects in a 2D CNA plot such as obtained from the 'plot.cna' and various 'layout' functions.
`labels`	An optional character vector giving labels for the points. Will be coerced using 'as.character', and recycled if necessary to the length of 'x'. Excess labels will be discarded, with a warning.
`cna`	A network object as returned from the 'cna' function.
`...`	Extra options passed to 'identify' function.

Details

This function calls the 'identify' and 'summary.cna' functions to query and label 2D CNA protein structure network plots produced by the 'plot.cna' function. Clicking with the mouse on plot points will add the corresponding labels and them to the plot and returned list object. A click with the right mouse button will stop the function.

Value

If 'labels' or 'cna' inputs are provided then a membership vector will be returned with the selected community ids and their members. Otherwise a vector with the ids of the selected communities will be returned.

Author(s)

Guido Scarabelli and Barry Grant

See Also

plot.cna, identify, plot.igraph, plot.communities, igraph.plotting

Examples

```
## Not run:

attach(hivp)

# Read the starting PDB file to determine atom correspondence
pdbfile <- system.file("examples/hivp.pdb", package="bio3d")
pdb <- read.pdb(pdbfile)

# Plot the network
xy <- plot.cna(net)

# Use identify.cna on the communities
d <- identify.cna(xy, cna=net)

# Right click to end the function...
## d <- identify(xy, summary(net)$members)

## End(Not run)
```

---

inner.prod *Mass-weighted Inner Product*

Description

Inner product of vectors (mass-weighted if requested).

Usage

```
inner.prod(x, y, mass=NULL)
```

Arguments

x	a numeric vector or matrix.
y	a numeric vector or matrix.
mass	a numeric vector containing the atomic masses for weighting.

Details

This function calculates the inner product between two vectors, or alternatively, the column-wise vector elements of matrices. If atomic masses are provided, the dot products will be mass-weighted.

See examples for more details.

Value

Returns the inner product(s).

Author(s)

Lars Skjaerven

References

Grant, B.J. et al. (2006) *Bioinformatics* **22**, 2695–2696.

See Also

nma , normalize.vector

Examples

```
## Matrix operations
x <- 1:3
y <- diag(x)
z <- matrix(1:9, ncol = 3, nrow = 3)

inner.prod(x,y)
inner.prod(y,z)


## Application to normal modes
pdb <- read.pdb( system.file("examples/1hel.pdb", package="bio3d") )

## Calculate (vibrational) normal modes
modes <- nma(pdb)

## Check for orthogonality
inner.prod(modes$U[,7], modes$U[,8])
```

---

inspect.connectivity *Check the Connectivity of Protein Structures*

Description

Investigate protein coordinates to determine if the structure has missing residues.

Usage

```
inspect.connectivity(pdbs, cut=4.)
```

Arguments

pdbs	an object of class 3daling as obtained from function pdbaln or read.fasta.pdb; a xyz matrix containing the cartesian coordinates of C-alpha atoms; or a 'pdb' object as obtained from function read.pdb.
cut	cutoff value to determine residue connectvitiy.

Details

Utility function for checking if the PDB structures in a 'pdbs' object contains missing residues inside the structure.

Value

Returns a vector.

Author(s)

Lars Skjaerven

References

Grant, B.J. et al. (2006) *Bioinformatics* **22**, 2695–2696.

See Also

dm, gap.inspect

Examples

```
## Not run:
## Fetch PDB files and split to chain A only PDB files
ids <- c("1a70_A", "1czp_A", "1frd_A", "1fxi_A", "1iue_A", "1pfd_A")
raw.files <- get.pdb(ids, path = "raw_pdbs")
files <- pdbsplit(raw.files, ids, path = "raw_pdbs/split_chain")

## Sequence Alignement, and connectivity check
pdbs <- pdbaln(files)

cons <- inspect.connectivity(pdbs)

## omit files with missing residues
files = files[cons]

## End(Not run)
```

`is.gap` *Gap Characters*

Description

Test for the presence of gap characters.

Usage

```
is.gap(x, gap.char = c("-", "."))
```

Arguments

`x`	an R object to be tested.
`gap.char`	a character vector containing the gap character types to test for.

Value

Returns a logical vector with the same length as the input 'x', with TRUE elements corresponding to 'gap.char' matches.

Note

During alignment, gaps are introduced into sequences that are believed to have undergone deletions or insertions with respect to other sequences in the alignment. These gaps, often referred to as indels, can be represented with 'NA', '-' or '.' characters.

This function provides a simple test for the presence of such characters, or indeed any set of user defined characters set by the 'gap.char' argument.

Author(s)

Barry Grant

References

Grant, B.J. et al. (2006) *Bioinformatics* **22**, 2695–2696.

See Also

`gap.inspect`, `read.fasta`, `read.fasta.pdb`, `seqaln`

Examples

```
is.gap( c("G",".","X","-","G","K","S","T") )

## Not run:
aln <- read.fasta( system.file("examples/kif1a.fa",
                   package = "bio3d") )

##- Mask existing gaps with an "X"
xaln <- aln
xaln$ali[ is.gap(xaln$ali) ]="X"

##- Read a new PDB and align its sequence to the existing alignment
pdb <- read.pdb( "1mkj" )
seq2aln(pdbseq(pdb), xaln, id = "1mkj")

## End(Not run)
```

---

is.pdb *Is an Object of Class 'pdb(s)'?*

Description

Checks whether its argument is an object of class 'pdb' or 'pdbs'.

Usage

```
is.pdb(x)
is.pdbs(x)
```

Arguments

x	an R object.

Details

Tests if the object 'x' is of class 'pdb' (`is.pdb`) or 'pdbs' (`is.pdbs`), i.e. if 'x' has a "class" attribute equal to pdb or pdbs.

Value

TRUE if x is an object of class 'pdb(s)' and FALSE otherwise

See Also

`read.pdb`, `read.fasta.pdb`, `pdbaln`

Examples

```
# Read a PDB file
pdb <- read.pdb( system.file("examples/1hel.pdb", package="bio3d") )
is.pdb(pdb)
```

`is.select` *Is an Object of Class 'select'?*

Description

Checks whether its argument is an object of class 'select'.

Usage

```
is.select(x)
```

Arguments

x	an R object to be tested.

Details

Tests if x is an object of class 'select', i.e. if x has a "class" attribute equal to `select`.

Value

TRUE if x is an object of class 'select' and FALSE otherwise

Author(s)

Julien Ide

See Also

`atom.select`

Examples

```
# Read a PDB file
pdb <- read.pdb( system.file("examples/1hel.pdb", package="bio3d") )

# Print structure summary
atom.select(pdb)

# Select all C-alpha atoms with residues numbers between 43 and 54
ca.inds <- atom.select(pdb, "calpha", resno=43:54)
is.select(ca.inds)
```

is.xyz *Is an Object of Class 'xyz'?*

Description

Checks whether its argument is an object of class 'xyz'.

Usage

```
is.xyz(x)
as.xyz(x)
```

Arguments

x	an R object to be tested

Details

Tests if x is an object of class 'xyz', i.e. if x has a "class" attribute equal to xyz.

Value

TRUE if x is an object of class 'xyz' and FALSE otherwise

See Also

read.pdb, read.ncdf, read.dcd, fit.xyz

Examples

```
# Read a PDB file
pdb <- read.pdb( system.file("examples/1hel.pdb", package="bio3d") )
is.xyz(pdb$xyz)
```

layout.cna *Protein Structure Network Layout*

Description

Determine protein structure network layout in 2D and 3D from the geometric center of each community.

Usage

```
layout.cna(x, pdb, renumber=TRUE, k=2, full=FALSE)
```

Arguments

x	A protein structure network object as obtained from the 'cna' function.
pdb	A pdb class object as obtained from the 'read.pdb' function.
renumber	Logical, if TRUE the input 'pdb' will be re-numbered starting at residue number one before community coordinate averages are calculated.
k	A single element numeric vector between 1 and 3 specifying the returned coordinate dimensions.
full	Logical, if TRUE the full all-Calpha atom network coordinates will be returned rather than the default clustered network community coordinates.

Details

This function calculates the geometric center for each community from the atomic position of it's Calpha atoms taken from a corresponding PDB file. Care needs to be taken to ensure the PDB residue numbers and the community vector names/length match.

The community residue membership are typically taken from the input network object but can be supplied as a list object with 'x$communities$membership'.

Value

A numeric matrix of Nxk, where N is the number of communities and k the number of dimensions requested.

Author(s)

Guido Scarabelli and Barry Grant

See Also

plot.cna, plot.communities, igraph.plotting, plot.igraph

Examples

```
# Load the correlation network
attach(hivp)

# Read the starting PDB file to determine atom correspondence
pdbfile <- system.file("examples/hivp.pdb", package="bio3d")
pdb <- read.pdb(pdbfile)

# Plot will be slow
xy <- plot.cna(net)
#plot3d.cna(net, pdb)

layout.cna(net, pdb, k=3)
layout.cna(net, pdb)

# can be used as input to plot.cna and plot3d.cna....
```

```
# plot.cna( net, layout=layout.cna(net, pdb) )
# plot3d.cna(net, pdb, layout=layout.cna(net, pdb, k=3))
```

---

lbio3d *List all Functions in the bio3d Package*

Description

A simple shortcut for ls("package:bio3d").

Usage

```
lbio3d()
```

Value

A character vector of function names from the bio3d package.

Author(s)

Barry Grant

References

Grant, B.J. et al. (2006) *Bioinformatics* **22**, 2695–2696.

---

lmi *LMI: Linear Mutual Information Matrix*

Description

Calculate the linear mutural information correlations of atomic displacements.

Usage

```
lmi(trj, grpby=NULL, ncore=1)
```

Arguments

trj	a numeric matrix of Cartesian coordinates with a row per structure/frame.
grpby	a vector counting connective duplicated elements that indicate the elements of trj that should be considered as a group (e.g. atoms from a particular residue).
ncore	number of CPU cores used to do the calculation. ncore>1 requires package 'parallel' installed.

Details

The correlation of the atomic fluctuations of a system can be assessed by the Linear Mutual Information (LMI) and the LMI has no unwanted dependency on the relative orientation of the fluctuations which the Pearson coefficient suffers from.

This function returns a matrix of all atom-wise linear mutual information whose elements are denoted as Cij. If Cij = 1, the fluctuations of atoms i and j are completely correlated and if Cij = 0, the fluctuations of atoms i and j are not correlated.

Value

Returns a linear mutual information matrix.

Author(s)

Hongyang Li & Barry Grant

References

Grant, B.J. et al. (2006) *Bioinformatics* **22**, 2695–2696. Lange, O.F. and Grubmuller, H. (2006) *PROTEINS: Structure, Function, and Bioinformatics* **62**:1053–1061.

Examples

```
##-- Read example trajectory file
trtfile <- system.file("examples/hivp.dcd", package="bio3d")
trj <- read.dcd(trtfile)

## Read the starting PDB file to determine atom correspondence
pdbfile <- system.file("examples/hivp.pdb", package="bio3d")
pdb <- read.pdb(pdbfile)

## select residues 24 to 27 and 85 to 90 in both chains
inds <- atom.select(pdb,"///24:27,85:90///CA/")

## lsq fit of trj on pdb
xyz <- fit.xyz(pdb$xyz, trj, fixed.inds=inds$xyz, mobile.inds=inds$xyz)

## LMI matrix (slow to run so restrict to Calpha)
cij <- lmi(xyz)

## Plot LMI matrix
#plot(cij)
col.scale <- colorRampPalette(c("gray95", "cyan"))(5)
plot(cij, at=seq(0.4,1, length=5), col.regions=col.scale)
```

load.enmff *ENM Force Field Loader*

Description

Load force field for elastic network normal mode calculation.

Usage

```
load.enmff(ff = 'calpha')
ff.calpha(r, rmin=2.9, ...)
ff.anm(r, cutoff=15, gamma=1, ...)
ff.pfanm(r, cutoff=NULL, ...)
ff.calphax(r, atom.id, ssdat=NULL, verbose=FALSE, ...)
ff.sdenm(r, atom.id, ssdat=NULL, ...)
ff.reach(r, atom.id, ssdat=NULL, ...)
```

Arguments

ff	a character string specifying the force field to use: 'calpha', 'anm', 'pfanm', 'calphax', 'reach', or 'sdenm'.
r	a numeric vector of c-alpha distances.
rmin	lowest allowed atom-atom distance for the force constant calculation. The default of 2.9A is based on an evaluation of 24 high-resolution X-ray structures (< 1A).
cutoff	numerical, cutoff for pair-wise interactions.
gamma	numerical, global scaling factor.
atom.id	atomic index.
ssdat	sequence and structure data.
verbose	logical, if TRUE interaction details are printed.
...	additional arguments (for technical reasons).

Details

This function provides a collection of elastic network model (ENM) force fields for normal modes analysis (NMA) of protein structures. It returns a function for calculating the residue-residue spring force constants.

The 'calpha' force field - originally developed by Konrad Hinsen - is the recommended one for most applications. It employs a spring force constant differentiating between nearest-neighbour pairs along the backbone and all other pairs. The force constant function was parameterized by fitting to a local minimum of a crambin model using the AMBER94 force field.

The implementation of the 'ANM' (Anisotropic Network Model) force field originates from the lab of Ivet Bahar. It uses a simplified (step function) spring force constant based on the pair-wise

distance. A variant of this from the Jernigan lab is the so-called 'pfANM' (parameter free ANM) with interactions that fall off with the square of the distance.

The 'calphax' force field is an extension of the original 'calpha' force field by Hinsen. In this implementation we have included specific force constants for disulfide bridges, helix 1-4 interactions, and beta sheet bridges.

The 'sdENM' (by Dehouck and Mikhailov) employs residue specific spring force constants. It has been parameterized through a statistical analysis of a total of 1500 NMR ensembles.

The 'REACH' force field (by Moritsugu and Smith) is parameterized based on variance-covariance matrices obtained from MD simulations. It employs force constants that fall off exponentially with distance for non-bonded pairs.

See references for more details on the individual force fields.

Value

'load.enmff' returns a function for calculating the spring force constants. The 'ff' functions returns a numeric vector of residue-residue spring force constants.

Note

The arguments 'atom.id' and 'ssdat' are used from within function 'build.hessian' for functions that are not simply a function of the pair-wise distance. e.g. the force constants in the 'calphax' model is a function of c-alpha distances, SSE information and SS-bonds, while the 'sdENM' force field computes the force constants based on a function of the residue types and calpha distance.

Author(s)

Lars Skjaerven

References

Hinsen, K. et al. (2000) *Chemical Physics* **261**, 25–37. Atilgan, A.R. et al. (2001) *Biophysical Journal* **80**, 505–515. Dehouck Y. & Mikhailov A.S. (2013) *PLoS Comput Biol* **9**:e1003209. Moritsugu K. & Smith J.C. (2008) *Biophysical Journal* **95**, 1639–1648. Yang, L. et al. (2009) *PNAS* **104**, 12347-52. Grant, B.J. et al. (2006) *Bioinformatics* **22**, 2695–2696.

See Also

nma, build.hessian

Examples

```
## Load the c-alpha force field
pfc.fun <- load.enmff('calpha')

## Calculate the pair force constant for a set of C-alpha distances
force.constants <- pfc.fun( seq(4,8, by=0.5) )

## Calculate the complete spring force constant matrix
## Fetch PDB
```

```
pdb <- read.pdb( system.file("examples/1hel.pdb", package="bio3d") )

## Fetch only c-alpha coordinates
ca.inds <- atom.select(pdb, 'calpha')
xyz <- pdb$xyz[ca.inds$xyz]

## Calculate distance matrix
dists <- dm.xyz(xyz, mask.lower=FALSE)

## all pair-wise spring force constants
fc.matrix <- apply(dists, 1, pfc.fun)
```

---

mktrj *PCA / NMA Atomic Displacement Trajectory*

Description

Make a trajectory of atomic displacments along a given principal component / normal mode.

Usage

```
mktrj(x, ...)
```

Arguments

x	a list object of class "pca" or "nma" as obtained with pca.xyz or nma, respectively.
...	additional arguments passed to the methods mktrj.pca or mktrj.nma.

Details

mktrj is a generic function calling the corresponding function determined by the class of the input argument x. Use methods("mktrj") to get all the methods for mktrj generic:

mktrj.pca will be used when x is an object of "pca".

mktrj.nma will be used when x is an object of "nma".

See examples for each corresponding function for more details.

Note

Molecular graphics software such as VMD or PyMOL is useful for viewing trajectories see e.g: http://www.ks.uiuc.edu/Research/vmd/.

Author(s)

Barry Grant, Lars Skjaerven

References

Grant, B.J. et al. (2006) *Bioinformatics* **22**, 2695–2696.

See Also

mktrj.pca, mktrj.nma, pca.xyz, nma, view.modes.

Examples

```
data(transducin)
attach(transducin, warn.conflicts=FALSE)

# Ignore gap containing positions
gaps.pos <- gap.inspect(pdbs$xyz)

# PCA
pc.xray <- pca.xyz(pdbs$xyz[, gaps.pos$f.inds])

# Write PC trajectory of pc=1
outfile = tempfile()
a <- mktrj(pc.xray, file = outfile)
outfile

detach(transducin)
```

---

mktrj.enma *Ensemble NMA Atomic Displacement Trajectory*

Description

Make a trajectory of atomic displacments along a given normal mode vector.

Usage

```
## S3 method for class 'enma'
mktrj(x = NULL, pdbs = NULL, s.inds = NULL, m.inds = NULL,
                     mag = 10, step = 1.25, file = NULL,
                     rock = TRUE, ncore = NULL, ...)
```

Arguments

x	a list object of class "enma" (obtained with nma.pdbs).
pdbs	a list object of class "pdbs" (obtained with pdbaln or read.fasta.pdb) which corresponds to the "enma" object.
s.inds	index or indices pointing to the structure(s) in the enma object for which the trajectory shall be generated.
m.inds	the mode number(s) along which displacements should be made.

mag	a magnification factor for scaling the displacements.
step	the step size by which to increment along the mode.
file	a character vector giving the output PDB file name.
rock	logical, if TRUE the trajectory rocks.
ncore	number of CPU cores used to do the calculation. ncore>1 requires package 'parallel' installed.
...	extra arguments to be passed to the function write.pdb.

Details

Trajectory frames are built from reconstructed Cartesian coordinates produced by interpolating from the structure along a given mode vector, in increments of step.

An optional magnification factor can be used to amplify displacements. This involves scaling the mode vector by mag-times.

Value

Returns a numeric matrix of interpolated coordinates with a row per structure.

Note

Molecular graphics software such as VMD or PyMOL is useful for viewing trajectories see e.g: http://www.ks.uiuc.edu/Research/vmd/.

Author(s)

Barry Grant, Lars Skjaerven

References

Grant, B.J. et al. (2006) *Bioinformatics* **22**, 2695–2696.

See Also

nma, view.modes, mktrj.nma, nma.pdbs, pdbaln

---

mktrj.nma *NMA Atomic Displacement Trajectory*

Description

Make a trajectory of atomic displacments along a given normal mode vector.

Usage

```
## S3 method for class 'nma'
mktrj(x = NULL, mode = 7, mag = 10, step = 1.25, file = NULL,  ...)
```

Arguments

x	a list object of class "nma" (obtained with nma).
mode	the mode number along which displacements should be made.
mag	a magnification factor for scaling the displacements.
step	the step size by which to increment along the mode.
file	a character vector giving the output PDB file name.
...	extra arguments to be passed to the function write.pdb.

Details

Trajectory frames are built from reconstructed Cartesian coordinates produced by interpolating from the structure along a given mode vector, in increments of step.

An optional magnification factor can be used to amplify displacements. This involves scaling the mode vector by mag-times.

Value

Returns a numeric matrix of interpolated coordinates with a row per structure.

Note

Molecular graphics software such as VMD or PyMOL is useful for viewing trajectories see e.g: http://www.ks.uiuc.edu/Research/vmd/.

Author(s)

Barry Grant, Lars Skjaerven

References

Grant, B.J. et al. (2006) *Bioinformatics* **22**, 2695–2696.

See Also

nma, view.modes.

Examples

```
## Fetch stucture
pdb <- read.pdb( system.file("examples/1hel.pdb", package="bio3d") )

## Calculate (vibrational) normal modes
modes <- nma(pdb)

## Visualize modes
outfile = file.path(tempdir(), "mode_7.pdb")
m7 <- mktrj(modes, mode=7, file = outfile)
outfile
```

mktrj.pca *PCA Atomic Displacement Trajectory*

Description

Make a trajectory of atomic displacments along a given principal component.

Usage

```
## S3 method for class 'pca'
mktrj(x = NULL, pc = 1, mag = 1, step = 0.125, file = NULL,  ...)
```

Arguments

x	a list object of class "pca" (obtained with pca.xyz).
pc	the PC number along which displacements should be made.
mag	a magnification factor for scaling the displacements.
step	the step size by which to increment along the pc.
file	a character vector giving the output PDB file name.
...	extra arguments to be passed to the function write.pdb.

Details

Trajectory frames are built from reconstructed Cartesian coordinates produced by interpolating from the mean structure along a given pc, in increments of step.

An optional magnification factor can be used to amplify displacements. This involves scaling by mag-times the standard deviation of the conformer distribution along the given pc (i.e. the square root of the associated eigenvalue).

Value

Returns a numeric matrix of interpolated coordinates with a row per structure.

Note

Molecular graphics software such as VMD or PyMOL is useful for viewing trajectories see e.g: http://www.ks.uiuc.edu/Research/vmd/.

Author(s)

Barry Grant

References

Grant, B.J. et al. (2006) *Bioinformatics* **22**, 2695–2696.

See Also

pca.xyz, view.modes.

Examples

```
data(transducin)
attach(transducin, warn.conflicts=FALSE)

# Ignore gap containing positions
gaps.res <- gap.inspect(pdbs$ali)
gaps.pos <- gap.inspect(pdbs$xyz)

# PCA
pc.xray <- pca.xyz(pdbs$xyz[, gaps.pos$f.inds])

# Write PC trajectory
path = tempdir()
a <- mktrj(pc.xray, pc=1, file=file.path(path, "pc1.pdb"),
           resno = pdbs$resno[1, gaps.res$f.inds],
           resid = aa123(pdbs$ali[1, gaps.res$f.inds]) )

b <- mktrj(pc.xray, pc=2, file=file.path(path, "pc2.pdb"),
           resno = pdbs$resno[1, gaps.res$f.inds],
           resid = aa123(pdbs$ali[1, gaps.res$f.inds]) )

c <- mktrj(pc.xray, pc=3, file=file.path(path, "pc3.pdb"),
           resno = pdbs$resno[1, gaps.res$f.inds],
           resid = aa123(pdbs$ali[1, gaps.res$f.inds]) )

# Output files
file.path(path, paste("pc", 1:3, ".pdb", sep=""))

detach(transducin)
```

---

motif.find *Find Sequence Motifs.*

Description

Return Position Indices of a Short Sequence Motif Within a Larger Sequence.

Usage

```
motif.find(motif, sequence)
```

Arguments

motif	a character vector of the short sequence motif.
sequence	a character vector of the larger sequence.

Details

The sequence and the motif can be given as a either a multiple or single element character vector. The dot character and other valid regexpr characters are allowed in the motif, see examples.

Value

Returns a vector of position indices within the sequence where the motif was found, see examples.

Author(s)

Barry Grant

References

Grant, B.J. et al. (2006) *Bioinformatics* **22**, 2695–2696.

See Also

regexpr, read.fasta, pdbseq

Examples

```
aa.seq <- pdbseq( read.pdb( get.pdb("4q21", URLonly=TRUE) ) )
motif = c("G....GKS")
motif.find(motif, aa.seq)
```

---

mustang *Structure-based Sequence Alignment with MUSTANG*

Description

Create a multiple sequence alignment from a bunch of PDB files.

Usage

```
mustang(files, exefile="mustang", outfile="aln.mustang.fa",
       verbose=TRUE)
```

Arguments

files	a character vector of PDB file names.
exefile	file path to the 'MUSTANG' program on your system (i.e. how is 'MUSTANG' invoked).
outfile	name of 'FASTA' output file to which alignment should be written.
verbose	logical, if TRUE 'MUSTANG' warning and error messages are printed.

Details

Structure-based sequence alignment with 'MUSTANG' attempts to arrange and align the sequences of proteins based on their 3D structure.

This function calls the 'MUSTANG' program, to perform a multiple sequence alignment, which MUST BE INSTALLED on your system and in the search path for executables.

Value

A list with two components:

ali	an alignment character matrix with a row per sequence and a column per equivalent aminoacid.
ids	sequence names as identifers.

Note

A system call is made to the 'MUSTANG' program, which must be installed on your system and in the search path for executables.

Author(s)

Lars Skjaerven

References

Grant, B.J. et al. (2006) *Bioinformatics* **22**, 2695–2696.

'MUSTANG' is the work of Konagurthu et al: Konagurthu, A.S. et al. (2006) *Proteins* **64**(3):559–74.

More details of the 'MUSTANG' algorithm, along with download and installation instructions can be obtained from:
http://www.csse.monash.edu.au/~karun/Site/mustang.html.

See Also

read.fasta, read.fasta.pdb, pdbaln, plot.fasta, seqaln

Examples

```
## Not run:
##-- Read a folder/directory of PDB files
#pdb.path <- "my_dir_of_pdbs"
#files  <- list.files(path=pdb.path ,
#                     pattern=".pdb",
#                     full.names=TRUE)

##-- Align these PDB sequences
aln <- mustang(files)

##-- Read Aligned PDBs storing coordinate data
pdbs <- read.fasta.pdb(aln)


## End(Not run)
```

`network.amendment` *Amendment of a CNA Network According To A Input Community Membership Vector.*

Description

This function changes the 'communities' attribute of a 'cna' class object to match a given membership vector.

Usage

```
network.amendment(x, membership, minus.log=TRUE)
```

Arguments

`x`	A protein network graph object as obtained from the 'cna' function.
`membership`	A numeric vector containing the new community membership.
`minus.log`	Logical. Whether to use the minus.log on the cij values.

Details

This function is useful, in combination with 'community.tree', for inspecting different community partitioning options of a input 'cna' object. See examples.

Value

Returns a 'cna' class object with the attributes changed according to the membership vector provided.

Author(s)

Guido Scarabelli

See Also

cna, community.tree, summary.cna

Examples

```
##-- Build a CNA object
pdb <- read.pdb("4Q21")
modes <- nma(pdb)
cij <- dccm(modes)
net <- cna(cij, cutoff.cij=0.2)

##-- Community membership vector for each clustering step
tree <- community.tree(net, rescale=TRUE)

## Produce a new k=7 membership vector and CNA network
memb.k7 <- tree$tree[ tree$num.of.comms == 7, ]
net.7 <- network.amendment(net, memb.k7)

plot(net.7, pdb)

print(net)
print(net.7)
```

---

nma *Normal Mode Analysis*

Description

Performs normal mode analysis (NMA) on either a single or an ensemble of protein structures

Usage

```
nma(...)
```

Arguments

...	arguments passed to the methods nma.pdb, or nma.pdbs. This will minimally include an object of class pdb as obtained from function read.pdb. Alternatively, an object of class pdbs as obtained from function pdbaln or read.fasta.pdb.

Details

nma is a generic function calling the corresponding function determined by the class of the input argument x. Use methods("nma") to get all the methods for nma generic:

nma.pdb will be used when x is of class pdb as obtained from function read.pdb.

nma.pdbs will perform 'ensemble' normal mode analysis of the PDB structures stored in the pdbs object.

See documentation and examples for each corresponding function for more details.

Author(s)

Lars Skjaerven

References

Grant, B.J. et al. (2006) *Bioinformatics* **22**, 2695–2696.

See Also

nma.pdb, nma.pdbs, pca.

---

nma.pdb *Normal Mode Analysis*

Description

Perform elastic network model (ENM) C-alpha normal modes calculation of a protein structure.

Usage

```
## S3 method for class 'pdb'
nma(pdb, inds = NULL, ff = 'calpha', pfc.fun = NULL,
                  mass = TRUE, temp = 300.0, keep = NULL, hessian = NULL,
                  outmodes = NULL, ... )

build.hessian(xyz, pfc.fun, fc.weights = NULL, sequ = NULL, sse = NULL,
    ss.bonds = NULL, ...)

## S3 method for class 'nma'
print(x, nmodes=6, ...)
```

Arguments

pdb	an object of class pdb as obtained from function read.pdb.
inds	atom and xyz coordinate indices obtained from atom.select that selects the elements of pdb upon which the calculation should be based. If not provided the function will attempt to select the calpha atoms automatically (based on function atom.select).
ff	character string specifying the force field to use: 'calpha', 'anm', 'pfanm', 'calphax', 'reach', or 'sdenm'.
pfc.fun	customized pair force constant ('pfc') function. The provided function should take a vector of distances as an argument to return a vector of force constants. If provided, 'pfc.fun' will override argument ff. See examples below.
mass	logical, if TRUE the Hessian will be mass-weighted.
temp	numerical, temperature for which the amplitudes for scaling the atomic displacement vectors are calculated. Set 'temp=NULL' to avoid scaling.
keep	numerical, final number of modes to be stored. Note that all subsequent analyses are limited to this subset of modes. This option is useful for very large structures and cases where memory may be limiting.
hessian	hessian matrix as obtained from build.hessian. For internal purposes and generally not intended for public use.
outmodes	atom indices as obtained from atom.select) specifying the atoms to include in the resulting mode object. For aanma, also character strings 'calpha' (which is the default) or 'noh' can be used.
xyz	a numeric vector of Cartesian coordinates.
fc.weights	a numeric matrix of size NxN (where N is the number of calpha atoms) containg scaling factors for the pariwise force constants. See examples below.
sse	secondary structure elements as obtained from dssp.
sequ	a character vector of the amino acid sequence.
ss.bonds	a numeric two-column matrix containing the residue numbers of the disulfide bridges in the structure.
x	an nma object obtained from nma.pdb.
nmodes	numeric, number of modes to be printed.
...	additional arguments to build.hessian, aa2mass, pfc.fun, and print. One useful option here for dealing with unconventional residues is 'mass.custom', see the aa2mass function for details.

Details

This function calculates the normal modes of a C-alpha model of a protein structure. A number of force fields are implemented all of whhich employ the elastic network model (ENM).

The 'calpha' force field - originally developed by Konrad Hinsen - is the recommended one for most applications. It employs a spring force constant differentiating between nearest-neighbour pairs along the backbone and all other pairs. The force constant function was parameterized by fitting to a local minimum of a crambin model using the AMBER94 force field.

See load.enmff for details of the different force fields.

By default nma.pdb() will diagonalize the mass-weighted Hessian matrix. The resulting mode vectors are moreover scaled by the thermal fluctuation amplitudes.

The implementation under default arguments reproduces the calculation of normal modes (VibrationalModes) in the Molecular Modeling Toolkit (MMTK) package. To reproduce ANM modes set ff='anm', mass=FALSE, and temp=NULL.

Value

Returns an object of class 'nma' with the following components:

modes	numeric matrix with columns containing the normal mode vectors. Mode vectors are converted to unweighted Cartesian coordinates when mass=TRUE. Note that the 6 first trivial eigenvectos appear in columns one to six.
frequencies	numeric vector containing the vibrational frequencies corresponding to each mode (for mass=TRUE).
force.constants	numeric vector containing the force constants corresponding to each mode (for mass=FALSE)).
fluctuations	numeric vector of atomic fluctuations.
U	numeric vector containing the raw eigenvectors. Equals to the modes component when mass=FALSE and temp=NULL.
L	numeric vector containing the raw eigenvalues.
xyz	numeric vector of the cartesian coordinates in which the calculation was performed.
mass	numeric vector containing the residue masses used for the mass-weighting.
temp	numerical, temperature for which the amplitudes for scaling the atomic displacement vectors are calculated.
triv.modes	number of trivial modes.
natoms	number of C-alpha atoms.
call	the matched call.

Note

The current version provides an efficent implementation of NMA with execution time comparable to similar software (when the entire Hessian is diagonalized).

The main (speed related) bottleneck is currently the diagonalization of the Hessian matrix which is performed with the core R function 'eigen'. For computing a few (5-20) approximate modes the user can consult package 'irlba'.

NMA is memory extensive and users should be cautions when running larger proteins (>3000 residues). Use 'keep' to reduce the amount of memory needed to store the final 'nma' object (the full 3Nx3N Hessian matrix still needs to be allocated).

We thank Edvin Fuglebakk for valuable discussions on the implementation as well as for contributing with testing.

Author(s)

Lars Skjaerven

References

Hinsen, K. et al. (2000) *Chemical Physics* **261**, 25–37. Grant, B.J. et al. (2006) *Bioinformatics* **22**, 2695–2696.

See Also

fluct.nma, mktrj.nma, dccm.nma, overlap, rmsip

Examples

```
## Fetch stucture
pdb <- read.pdb( system.file("examples/1hel.pdb", package="bio3d") )

## Calculate normal modes
modes <- nma(pdb)

## Print modes
print(modes)

## Plot modes
plot(modes)

## Visualize modes
#m7 <- mktrj.nma(modes, mode=7, file="mode_7.pdb")

## Not run:
## Use Anisotropic Network Model
modes <- nma(pdb, ff="anm", mass=FALSE, temp=NULL, cutoff=15)

## Use SSE information and SS-bonds
sse      <- dssp(pdb, resno=FALSE, full=TRUE)
ss.bonds <- matrix(c(76,94, 64,80, 30,115, 6,127),
                   ncol=2, byrow=TRUE)

modes <- nma(pdb, ff="calphax", sse=sse, ss.bonds=ss.bonds)


## User defined energy function
## Note: Must take a vector of distances
"my.ff" <- function(r) {
   ifelse( r>15, 0, 1 )
}

## Modes with a user defined energy function
modes <- nma(pdb, pfc.fun=my.ff)


## A more manual approach
```

```
sele <- atom.select(pdb, "//A////CA/")
xyz <- pdb$xyz[sele$xyz]

hessian <- build.hessian(xyz, my.ff)
modes <- eigen(hessian)

## Dealing with unconventional residues
pdb <- read.pdb("1xj0")
## nma(pdb)
modes <- nma(pdb, mass.custom=list(CSX=121.166))

## End(Not run)
```

`nma.pdbs` *Ensemble Normal Mode Analysis*

Description

Perform NMA on an ensemble of aligned protein structures.

Usage

```
## S3 method for class 'pdbs'
nma(pdbs, fit = TRUE, full = FALSE, subspace = NULL,
         rm.gaps = TRUE, varweight=FALSE,
         outpath = NULL, ncore = 1, ...)

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

Arguments

`pdbs`	a numeric matrix of aligned C-alpha xyz Cartesian coordinates. For example an alignment data structure obtained with `read.fasta.pdb` or `pdbaln`.
`fit`	logical, if TRUE coordinate superposition is performed prior to normal mode calculations.
`full`	logical, if TRUE return the complete, full structure, 'nma' objects.
`subspace`	number of eigenvectors to store for further analysis.
`rm.gaps`	logical, if TRUE obtain the hessian matrices for only atoms in the aligned positions (non-gap positions in all aligned structures). Thus, gap positions are removed from output.
`varweight`	logical, if TRUE perform weighing of the pair force constants. Alternatively, provide a NxN matrix containing the weights. See function `var.xyz`.
`outpath`	character string specifing the output directory to which the PDB structures should be written.

ncore	number of CPU cores used to do the calculation. ncore>1 requires package 'parallel' installed.
x	an enma object obtained from nma.pdbs.
...	additional arguments to nma, aa2mass, and print.

Details

This function performs normal mode analysis (NMA) on a set of aligned protein structures obtained with function read.fasta.pdb or pdbaln. The main purpose is to provide aligned atomic fluctuations and mode vectors in an automated fashion.

The normal modes are calculated on the full structures as provided by object 'pdbs'. With the input argument 'full=TRUE' the full 'nma' objects are returned together with output 'U.subs' providing the aligned mode vectors. When 'rm.gaps=TRUE' the unaligned atoms are ommited from output. With default arguments 'rmsip' provides RMSIP values for all pairwise structures.

See examples for more details.

Value

Returns an 'enma' object with the following components:

fluctuations	a numeric matrix with aligned atomic fluctuations.
rmsip	a numeric matrix of RMSIP values between all pairs of mode subsets.
U.subspace	a three-dimensional array with aligned eigenvectors (corresponding to the subspace defined by the first N non-trivial eigenvectors ('U') of the 'nma' object).
L	numeric matrix containing the raw eigenvalues.
xyz	an object of class 'xyz' containing the Cartesian coordinates in which the calculation was performed. Coordinates are superimposed to the first structure of the pdbs object when 'fit=TRUE'.
full.nma	a list with a nma object for each input structure.

Author(s)

Lars Skjaerven

References

Grant, B.J. et al. (2006) *Bioinformatics* **22**, 2695–2696.

See Also

For normal mode analysis on single structure PDB: nma.pdb

For the analysis of the resulting 'eNMA' object: mktrj.enma, dccm.enma, plot.enma, cov.enma.

Similarity measures: sip, covsoverlap, bhattacharyya, rmsip.

Related functionality: pdbaln, read.fasta.pdb.

Examples

```
## NOTE: FOLLOWING EXAMPLE NEEDS MUSCLE INSTALLED
if(check.utility("muscle")) {

## Fetch PDB files and split to chain A only PDB files
ids <- c("1a70_A", "1czp_A", "1frd_A", "1fxi_A", "1iue_A", "1pfd_A")
files <- get.pdb(ids, split = TRUE, path = tempdir())

## Sequence Alignement
pdbs <- pdbaln(files, outfile = tempfile())

## Normal mode analysis on aligned data
modes <- nma(pdbs, rm.gaps=FALSE)

## Plot fluctuation data
plot(modes, pdbs=pdbs)

## Cluster on Fluctuation similariy
sip <- sip(modes)
hc <- hclust(dist(sip))
col <- cutree(hc, k=3)

## Plot fluctuation data
plot(modes, pdbs=pdbs, col=col)

## Remove gaps from output
modes <- nma(pdbs, rm.gaps=TRUE)

## RMSIP is pre-calculated
heatmap(1-modes$rmsip)

## Bhattacharyya coefficient
bc <- bhattacharyya(modes)
heatmap(1-bc)

}
```

---

normalize.vector *Mass-Weighted Normalized Vector*

Description

Normalizes a vector (mass-weighted if requested).

Usage

```
normalize.vector(x, mass=NULL)
```

Arguments

x	a numeric vector or matrix to be normalized.
mass	a numeric vector containing the atomic masses for weighting.

Details

This function normalizes a vector, or alternatively, the column-wise vector elements of a matrix. If atomic masses are provided the vector is mass-weigthed.

See examples for more details.

Value

Returns the normalized vector(s).

Author(s)

Lars Skjaerven

References

Grant, B.J. et al. (2006) *Bioinformatics* **22**, 2695–2696.

See Also

nma , inner.prod

Examples

```
x <- 1:3
y <- matrix(1:9, ncol = 3, nrow = 3)

normalize.vector(x)
normalize.vector(y)

## Application to normal modes
pdb <- read.pdb( system.file("examples/1hel.pdb", package="bio3d") )

## Calculate (vibrational) normal modes
modes <- nma(pdb)

## Returns a vector
nv <- normalize.vector(modes$modes[,7])

## Returns a matrix
nv <- normalize.vector(modes$modes[,7:10])

## Mass-weighted
nv <- normalize.vector(modes$modes[,7], mass=modes$mass)
```

orient.pdb *Orient a PDB Structure*

Description

Center, to the coordinate origin, and orient, by principal axes, the coordinates of a given PDB structure or xyz vector.

Usage

```
orient.pdb(pdb, atom.subset = NULL, verbose = TRUE)
```

Arguments

pdb	a pdb data structure obtained from read.pdb or a vector of 'xyz' coordinates.
atom.subset	a subset of atom positions to base orientation on.
verbose	print dimension details.

Value

Returns a numeric vector of re-oriented coordinates.

Note

Centering and orientation can be restricted to a atom.subset of atoms.

Author(s)

Barry Grant

References

Grant, B.J. et al. (2006) *Bioinformatics* **22**, 2695–2696.

See Also

read.pdb, write.pdb, fit.xyz, rot.lsq , atom.select

Examples

```
pdb <- read.pdb( "1bg2" )
xyz <- orient.pdb(pdb)
#write.pdb(pdb, xyz = xyz, file = "mov1.pdb")


# Based on C-alphas
inds <- atom.select(pdb, "calpha")
```

```
xyz  <- orient.pdb(pdb, atom.subset=inds$atom)
#write.pdb(pdb, xyz = xyz, file = "mov2.pdb")


# Based on a central Beta-strand
inds <- atom.select(pdb, "///224:232///CA/")
xyz  <- orient.pdb(pdb, atom.subset=inds$atom)
#write.pdb(pdb, xyz = xyz, file = "mov3.pdb")
```

`overlap` *Overlap analysis*

Description

Calculate the Squared Overlap between sets of vectors.

Usage

```
overlap(modes, dv, nmodes=20)
```

Arguments

`modes`	an object of class `"pca"` or `"nma"` as obtained from function `pca.xyz` or `nma`. Alternatively a 3NxM matrix of eigenvectors can be provided.
`dv`	a displacement vector of length 3N.
`nmodes`	the number of modes in which the calculation should be based.

Details

Squared overlap (or dot product) is used to measure the similiarity between a displacement vector (e.g. a difference vector between two conformational states) and mode vectors obtained from principal component or normal modes analysis.

By definition the cumulative sum of the overlap values equals to one.

Structure `modes$U` (or alternatively, the 3NxM matrix of eigenvectors) should be of same length (3N) as `dv`.

Value

Returns a list with the following components:

`overlap`	a numeric vector of the squared dot products (overlap values) between the (normalized) vector (`dv`) and each mode in `mode`.
`overlap.cum`	a numeric vector of the cumulative squared overlap values.

Author(s)

Lars Skjaerven

References

Skjaerven, L. et al. (2011) *Proteins* **79**, 232–243. Grant, B.J. et al. (2006) *Bioinformatics* **22**, 2695–2696.

See Also

rmsip, pca.xyz, nma, difference.vector

Examples

```
data(kinesin)
attach(kinesin, warn.conflicts=FALSE)

# Ignore gap containing positions
##gaps.res <- gap.inspect(pdbs$ali)
gaps.pos <- gap.inspect(pdbs$xyz)

#-- Do PCA
pc.xray <- pca.xyz(pdbs$xyz[, gaps.pos$f.inds])

# Define a difference vector between two structural states
diff.inds <- c(grep("d1v8ja", pdbs$id),
               grep("d1goja", pdbs$id))

dv <- difference.vector( pdbs$xyz[diff.inds,], gaps.pos$f.inds )

# Calculate the squared overlap between the PCs and the difference vector
o <- overlap(pc.xray, dv)
o <- overlap(pc.xray$U, dv)

# Plot results
plot(o$overlap, type='h', ylim=c(0,1))
points(o$overlap)
lines(o$overlap.cum, type='b', col='red')


detach(kinesin)

## Not run:
## Calculate overlap from NMA
pdb.a <- read.pdb("1cmk")
pdb.b <- read.pdb("3dnd")

## Fetch CA coordinates
sele.a <- atom.select(pdb.a, "//E/15:350///CA")
sele.b <- atom.select(pdb.b, "//A/1:350///CA")

xyz <- rbind(pdb.a$xyz[sele.a$xyz],
             pdb.b$xyz[sele.b$xyz])

## Superimpose
xyz[2,] <- fit.xyz(xyz[1,], xyz[2,], 1:ncol(xyz))
```

```
## The difference between the two conformations
dv <- difference.vector( xyz )

## Calculate normal modes
modes <- nma(pdb.a, inds=sele.a)

# Calculate the squared overlap between the normal modes
# and the difference vector
o <- overlap(modes, dv)

## End(Not run)
```

`pairwise` *Pair Indices*

Description

A utility function to determine indices for pairwise comparisons.

Usage

```
pairwise(N)
```

Arguments

`N`	a single numeric value representing the total number of things to undergo pairwise comparison.

Value

Returns a two column numeric matrix giving the indices for all pairs.

Author(s)

Barry Grant

References

Grant, B.J. et al. (2006) *Bioinformatics* **22**, 2695–2696.

See Also

`seqidentity`

Examples

```
pairwise(3)
pairwise(20)
```

pca *Principal Component Analysis*

Description

Performs principal components analysis (PCA) on biomolecular structure data.

Usage

```
pca(...)
```

Arguments

...	arguments passed to the methods pca.xyz, pca.pdbs, etc. Typically this includes either a numeric matrix of Cartesian coordinates with a row per structure/frame (function pca.xyz()), or an object of class pdbs as obtained from function pdbaln or read.fasta.pdb (function pca.pdbs()).

Details

Principal component analysis can be performed on any structure dataset of equal or unequal sequence composition to capture and characterize inter-conformer relationships.

This generic pca function calls the corresponding methods function for actual calculation, which is determined by the class of the input argument x. Use methods("pca") to list all the current methods for pca generic. These will include:

pca.xyz, which will be used when x is a numeric matrix containing Cartesian coordinates (e.g. trajectory data).

pca.pdbs, which will perform PCA on the Cartesian coordinates of a input pdbs object (as obtained from the 'read.fasta.pdb' or 'pdbaln' functions).

Currently, function pca.tor should be called explicitly as there are currently no defined 'tor' object classes.

See the documentation and examples for each individual function for more details and worked examples.

Author(s)

Barry Grant, Lars Skjaerven

References

Grant, B.J. et al. (2006) *Bioinformatics* **22**, 2695–2696.

See Also

pca.xyz, pca.pdbs, pdbaln.

pca.array *Principal Component Analysis of an arrary of matrices*

Description

Calculate the principal components of an array of correlation or covariance matrices.

Usage

```
## S3 method for class 'array'
pca(x, use.svd = TRUE, ...)
```

Arguments

x	an array of matrices, e.g. correlation or covariance matrices as obtained from functions dccm or enma2covs.
use.svd	logical, if TRUE singular value decomposition (SVD) is called instead of eigenvalue decomposition.
...	.

Details

This function performs an PCA of residue-residue cross-correlations or covariance matrices derived from ensemble NMA of M structures.

Value

Returns a list with components equivalent to the output from pca.xyz.

Author(s)

Xin-Qiu Yao, Lars Skjaerven

References

Grant, B.J. et al. (2006) *Bioinformatics* **22**, 2695–2696.

See Also

pca.xyz

pca.pdbs *Principal Component Analysis*

Description

Performs principal components analysis (PCA) on an ensemble of PDB structures.

Usage

```
## S3 method for class 'pdbs'
pca(pdbs, core.find = FALSE, fit = FALSE, ...)
```

Arguments

pdbs	an object of class pdbs as obtained from function pdbaln or read.fasta.pdb.
core.find	logical, if TRUE core.find() function will be called to find core positions and coordinates of PDB structures will be fitted based on cores.
fit	logical, if TRUE coordinates of PDB structures will be fitted based on all CA atoms.
...	additional arguments passed to the method pca.xyz.

Details

The function pca.pdbs is a wrapper for the function pca.xyz, wherein more details of the PCA procedure are documented.

Value

Returns a list with the following components:

L	eigenvalues.
U	eigenvectors (i.e. the variable loadings).
z.u	scores of the supplied data on the pcs.
sdev	the standard deviations of the pcs.
mean	the means that were subtracted.

Author(s)

Barry Grant, Lars Skjaerven and Xin-Qiu Yao

References

Grant, B.J. et al. (2006) *Bioinformatics* **22**, 2695–2696.

See Also

pca, pca.xyz, pdbaln, nma.

Examples

```
attach(transducin)

#-- Do PCA ignoring gap containing positions
pc.xray <- pca(pdbs)

# Plot results (conformer plots & scree plot)
plot(pc.xray, col=annotation[, "color"])
```

`pca.tor` *Principal Component Analysis*

Description

Performs principal components analysis (PCA) on torsion angle `data`.

Usage

```
## S3 method for class 'tor'
pca(data, ...)
```

Arguments

`data`	numeric matrix of torsion angles with a row per structure.
`...`	additional arguments passed to the method `pca.xyz`.

Value

Returns a list with the following components:

`L`	eigenvalues.
`U`	eigenvectors (i.e. the variable loadings).
`z.u`	scores of the supplied `data` on the pcs.
`sdev`	the standard deviations of the pcs.
`mean`	the means that were subtracted.

Author(s)

Barry Grant and Karim ElSawy

References

Grant, B.J. et al. (2006) *Bioinformatics* **22**, 2695–2696.

See Also

torsion.xyz, plot.pca, plot.pca.loadings, pca.xyz

Examples

```
##-- PCA on torsion data for multiple PDBs
data(kinesin)
attach(kinesin, warn.conflicts=FALSE)

gaps.pos <- gap.inspect(pdbs$xyz)
tor <- t(apply( pdbs$xyz[, gaps.pos$f.inds], 1, torsion.xyz, atm.inc=1))
pc.tor <- pca.tor(tor[,-c(1,218,219,220)])
#plot(pc.tor)
plot.pca.loadings(pc.tor)

detach(kinesin)

## Not run:
##-- PCA on torsion data from an MD trajectory
trj <- read.dcd( system.file("examples/hivp.dcd", package="bio3d") )
tor <- t(apply(trj, 1, torsion.xyz, atm.inc=1))
gaps <- gap.inspect(tor)
pc.tor <- pca.tor(tor[,gaps$f.inds])
plot.pca.loadings(pc.tor)

## End(Not run)
```

---

pca.xyz *Principal Component Analysis*

Description

Performs principal components analysis (PCA) on a xyz numeric data matrix.

Usage

```
## S3 method for class 'xyz'
pca(xyz, subset = rep(TRUE, nrow(as.matrix(xyz))),
                 use.svd = FALSE, rm.gaps=FALSE, ...)

## S3 method for class 'pca'
print(x, nmodes=6, ...)
```

Arguments

xyz	numeric matrix of Cartesian coordinates with a row per structure.
subset	an optional vector of numeric indices that selects a subset of rows (e.g. experimental structures vs molecular dynamics trajectory structures) from the full xyz matrix. Note: the full xyz is projected onto this subspace.

use.svd	logical, if TRUE singular value decomposition (SVD) is called instead of eigenvalue decomposition.
rm.gaps	logical, if TRUE gap positions (with missing coordinate data in any input structure) are removed before calculation. This is equivalent to removing NA cols from xyz.
x	an object of class pca, as obtained from function pca.xyz.
nmodes	numeric, number of modes to be printed.
...	additional arguments to print.

Value

Returns a list with the following components:

L	eigenvalues.
U	eigenvectors (i.e. the x, y, and z variable loadings).
z	scores of the supplied xyz on the pcs.
au	atom-wise loadings (i.e. xyz normalized eigenvectors).
sdev	the standard deviations of the pcs.
mean	the means that were subtracted.

Author(s)

Barry Grant

References

Grant, B.J. et al. (2006) *Bioinformatics* **22**, 2695–2696.

See Also

pca, pca.pdbs, plot.pca, mktrj.pca, pca.tor, project.pca

Examples

```
## Not run:
#-- Read transducin alignment and structures
aln <- read.fasta(system.file("examples/transducin.fa",package="bio3d"))
pdbs <- read.fasta.pdb(aln)

# Find core
core <- core.find(pdbs,
                  #write.pdbs = TRUE,
                  verbose=TRUE)

rm(list=c("pdbs", "core"))

## End(Not run)
```

```
#-- OR for demo purposes just read previously saved transducin data
attach(transducin)

# Previously fitted coordinates based on sub 1.0A^3 core. See core.find() function.
xyz <- pdbs$xyz

#-- Do PCA ignoring gap containing positions
pc.xray <- pca.xyz(xyz, rm.gaps=TRUE)

# Plot results (conformer plots & scree plot)
plot(pc.xray, col=annotation[, "color"])

## Plot atom wise loadings
plot.bio3d(pc.xray$au[,1], ylab="PC1 (A)")


## Plot loadings in relation to reference structure 1TAG
gaps.res <- gap.inspect(pdbs$ali)
pdb <- read.pdb("1tag")

ind <- grep("1TAG", pdbs$id)
res.ind <- pdbs$resno[ind, gaps.res$f.ind]
op <- par(no.readonly=TRUE)
par(mfrow = c(3, 1), cex = 0.6, mar = c(3, 4, 1, 1))
plot.bio3d(res.ind, pc.xray$au[,1], sse=pdb, ylab="PC1 (A)")
plot.bio3d(res.ind, pc.xray$au[,2], sse=pdb, ylab="PC2 (A)")
plot.bio3d(res.ind, pc.xray$au[,3], sse=pdb, ylab="PC3 (A)")
par(op)


## Not run:
# Write PC trajectory
a <- mktrj.pca(pc.xray, pc=1, file="pc1.pdb",
               resno = pdbs$resno[1, gaps.res$f.inds],
               resid = aa123(pdbs$ali[1, gaps.res$f.inds]) )

b <- mktrj.pca(pc.xray, pc=2, file="pc2.pdb",
               resno = pdbs$resno[1, gaps.res$f.inds],
               resid = aa123(pdbs$ali[1, gaps.res$f.inds]) )

c <- mktrj.pca(pc.xray, pc=3, file="pc3.pdb",
               resno = pdbs$resno[1, gaps.res$f.inds],
               resid = aa123(pdbs$ali[1, gaps.res$f.inds]) )

## End(Not run)

detach(transducin)
```

---

pdb.annotate *Get Customizable Annotations From PDB*

Description

Get customizable annotations for query results from PDB.

Usage

```
pdb.annotate(ids, anno.terms = NULL, unique = FALSE, verbose = FALSE)
```

Arguments

ids	A charater vector of one or more 4-letter PDB codes/identifiers of the files for query.
anno.terms	Terms can be used for query. The "anno.terms" can be "structureId", "experimentalTechnique", "resolution", "chainId", "ligandId", "ligandName", "source", "scopDomain", "classification", "compound","title", "citation", "citationAuthor", "journalName", "publicationYear". If anno.terms=NULL, all information would be returned.
unique	logical, if TRUE only unique PDB entries are returned. Alternatively data for each chain ID is provided.
verbose	logical, if TRUE details of the RCurl postForm routine is printed.

Details

Given a list of PDB IDs and query terms, this function will download the required information from PDB, and return a data frame of query results.

Value

Returns a data frame of query results with a row for each PDB record, and annotation terms column-wise.

Author(s)

Hongyang Li, Barry Grant, Lars Skjaerven

Examples

```
# Fetch all annotation terms
ids <- c("6Q21_B", "1NVW", "1P2U_A")
anno <- pdb.annotate(ids)

# Access terms, e.g. ligand names:
anno$ligandName

## only unique PDB IDs
anno <- pdb.annotate(ids, unique=TRUE)

# Fetch only specific terms
```

```
pdb.annotate(ids, anno.terms = c("ligandId", "citation"))
```

---

pdb2aln *Align a PDB structure to an existing alignment*

Description

Extract the sequence from a PDB file and align it to an existing multiple sequence alignment that you wish keep intact.

Usage

```
pdb2aln(aln, pdb, id="seq.pdb", aln.id=NULL, exefile="muscle", file="pdb2aln.fa")
```

Arguments

aln	an alignment list object with id and ali components, similar to that generated by read.fasta, read.fasta.pdb, and seqaln.
pdb	the PDB object.
id	name for the PDB sequence in the new alignment.
aln.id	id of the sequence in the original alignment that is closest to the sequence of the PDB structure.
exefile	file path to the 'MUSCLE' program on your system (i.e. how is 'MUSCLE' invoked).
file	file name for outputing the new alignment.

Details

This function aligns a PDB sequence to an alignment and stores the mappings between the new and existing alignments, as well as the mappings between new alignment and the PDB atomic indices.

The function can be used to perform the routine procedure of finding the indices of CA atoms in the PDB structure, the residue numbers of which are equivalent to the predefined positions in the existing alignment. For example, when we project a MD simulation trajectory onto the low dimensional subspace derived from the PCA of cystallographic structures, we need first align the sequence of the simulated protein to the original alignment of crystal structures (or find out the identical sequence in the alignment if the simulation started from one of the crystal structures). Then residues of the simulation system equivalent to those used for fitting crystal structures and performing PCA can be identified. The corresponding CA atoms to be used for fitting and projecting the trajectory are then obtained by mapping the equivalent residues onto the topology of the trajectory.

When aln.id is provided, the function will do pairwise alignment between the PDB sequence and the sequence in the alignment aln with id containing aln.id. This is the best way to use the function if the simulated protein has an identical or very similar sequence to one of the sequences in the alignment aln.

Value

Return a list object with three components:

id	sequence names as identifers.
ali	an alignment character matrix with a row per sequence and a column per equivalent aminoacid/nucleotide.
ref	an integer matrix with the first row the indices of original alignment and the second CA indices of the PDB structure.

Author(s)

Xin-Qiu Yao & Barry Grant

References

Grant, B.J. et al. (2006) *Bioinformatics* **22**, 2695–2696.

See Also

seq2aln, seqaln.pair, pdb2aln.ind

Examples

```
## Not run:
##--- Read aligned PDB coordinates (CA only)
aln  <- read.fasta(system.file("examples/kif1a.fa",package="bio3d"))
pdbs <- read.fasta.pdb(aln)

##--- Read PDB coordinate for a new structure (all atoms)
id <- get.pdb("2kin", URLonly=TRUE)
pdb <- read.pdb(id)

# map the non-gap positions
gap.inds <- gap.inspect(pdbs$resno)
naln <- pdb2aln(aln=pdbs, pdb=pdb, id=id)
ninds <- which(naln$ref["ali.pos", ] %in% gap.inds$f.inds)
npc.inds <- naln$ref["ca.inds", ninds]

# If gaps are found in PDB sequence with the predefined indices,
# redefine the non-gap positions
ngap.f.inds <- gap.inds$f.inds[!is.na(npc.inds)]
npc.inds <- npc.inds[!is.na(npc.inds)]

##--- fit the atomic coordinates to the aligned X-ray structure
xyz <- fit.xyz(pdbs$xyz[1,], pdb$xyz, atom2xyz(ngap.f.inds), atom2xyz(npc.inds))

## seq2aln(pdbseq(pdb), aln, id = id)
## do we get the same result


## End(Not run)
```

pdb2aln.ind *Mapping between PDB atomic indices and alignment positions*

Description

Find the best alignment between a PDB structure and an existing alignment. Then, given a set of residue indices defined for the original alignment, return the equivalent CA atom indices in the PDB coordinates.

Usage

```
pdb2aln.ind(aln, pdb, inds, ...)
```

Arguments

aln	an alignment list object with id and ali components, similar to that generated by read.fasta, read.fasta.pdb, and seqaln.
pdb	the PDB object.
inds	a numeric vector with a subset of the alignment position indices.
...	additional arguments for the function pdb2aln.

Details

Call the function pdb2aln to align the PDB sequence to the existing alignment. Then find the equivalent CA atomic indices in PDB to inds.

Value

Returns a numeric vector with the equivalent CA atomic indices.

Author(s)

Xin-Qiu Yao & Barry Grant

References

Grant, B.J. et al. (2006) *Bioinformatics* **22**, 2695–2696.

See Also

seq2aln, seqaln.pair, pdb2aln

Examples

```
## Not run:
##--- Read aligned PDB coordinates (CA only)
aln  <- read.fasta(system.file("examples/kif1a.fa",package="bio3d"))
pdbs <- read.fasta.pdb(aln)

##--- Read PDB coordinate for a new structure (all atoms)
id <- get.pdb("2kin", URLonly=TRUE)
pdb <- read.pdb(id)

# map the non-gap positions
gap.inds <- gap.inspect(pdbs$resno)
npc.inds <- pdb2aln.ind(aln=pdbs, pdb=pdb, id=id, inds=gap.inds$f.inds)

# If gaps are found in PDB sequence with the predefined indices,
# redefine the non-gap positions
ngap.f.inds <- gap.inds$f.inds[!is.na(npc.inds)]
npc.inds <- npc.inds[!is.na(npc.inds)]

##--- fit the atomic coordinates to the aligned X-ray structure
xyz <- fit.xyz(pdbs$xyz[1,], pdb$xyz, atom2xyz(ngap.f.inds), atom2xyz(npc.inds))

## seq2aln(pdbseq(pdb), aln, id = id)
## do we get the same result


## End(Not run)
```

`pdbaln` *Sequence Alignment of PDB Files*

Description

Create multiple sequences alignments from a list of PDB files returning aligned sequence and structure records.

Usage

```
pdbaln(files, fit = FALSE, pqr = FALSE, ncore = 1, nseg.scale = 1,  ...)
```

Arguments

`files`	a character vector of PDB file names.
`fit`	logical, if TRUE coordinate superposition is performed on the input structures.
`pqr`	logical, if TRUE the input structures are assumed to be in PQR format.
`ncore`	number of CPU cores used to do the calculation. `ncore>1` requires package 'parallel' installed.

nseg.scale	split input data into specified number of segments prior to running multiple core calculation. See fit.xyz.
...	extra arguments passed to seqaln function.

Details

This wrapper function calls the underlying functions read.pdb, pdbseq, seqaln and read.fasta.pdb returning a list of class "pdbs" similar to that returned by read.fasta.pdb.

As these steps are often error prone it is recomended for most cases that the individual underlying functions are called in sequence with checks made on the valadity of their respective outputs to ensure sensible results.

Value

Returns a list of class "pdbs" with the following five components:

xyz	numeric matrix of aligned C-alpha coordinates.
resno	character matrix of aligned residue numbers.
b	numeric matrix of aligned B-factor values.
chain	character matrix of aligned chain identifiers.
id	character vector of PDB sequence/structure names.
ali	character matrix of aligned sequences.
call	the matched call.

Note

See recommendation in details section above.

Author(s)

Barry Grant

References

Grant, B.J. et al. (2006) *Bioinformatics* **22**, 2695–2696.

See Also

read.pdb, pdbseq, seqaln, read.fasta,read.fasta.pdb, core.find, fit.xyz, read.all

Examples

```
## Not run:
#files <- get.pdb(c("4q21","5p21"), URLonly=TRUE)
files <- get.pdb(c("4q21","5p21"), path=tempdir(), overwrite=TRUE)
pdbaln(files)

## End(Not run)
```

pdbfit *PDB File Coordinate Superposition*

Description

Protein Databank Bank file coordinate superposition with the Kabsch algorithm.

Usage

```
pdbfit(pdbs, inds = NULL, outpath = NULL, ...)
```

Arguments

pdbs	a list of class "pdbs" containing PDB file data, as obtained from read.fasta.pdb or pdbaln.
inds	a vector of indices that selects the coordinate positions, in terms of x, y and z elements, upon which fitting should be based. This defults to all equivalent non-gap positions.
outpath	character string specifing the output directory for optional coordinate file output. Note that full files (i.e. all atom files) are written, seebelow.
...	extra arguments passed to fit.xyz function.

Details

The function pdbfit is a wrapper for the function fit.xyz, wherein full details of the superposition procedure are documented.

Input to this function should be a list object obtained with the function read.fasta.pdb or pdbaln. See the examples below.

The reference frame for supperposition (i.e. the fixed structure to which others are superposed) is the first entry in the input "pdbs" object. For finer control use fit.xyz.

Value

Returns moved coordinates.

Author(s)

Barry Grant

References

Grant, B.J. et al. (2006) *Bioinformatics* **22**, 2695–2696.

Kabsch *Acta Cryst* (1978) **A34**, 827–828.

See Also

pdbaln, read.fasta.pdb, fit.xyz, rmsd, read.pdb

Examples

```
## Not run:
#files <- get.pdb(c("4q21","5p21"), URLonly=TRUE)
files <- get.pdb(c("4q21","5p21"), path=tempdir(), overwrite=TRUE)
pdbs <- pdbaln(files)
xyz <- pdbfit(pdbs)

# Superpose again this time outputing PDBs
#xyz <- pdbaln( files, outpath="fitted" )

## End(Not run)
```

`pdbs.filter` *Filter or Trim a pdbs PDBs Object*

Description

Trim residues and/or filter out structures from a pdbs PDBs object.

Usage

```
pdbs.filter(pdbs, row.inds=NULL, col.inds=NULL)
```

Arguments

`pdbs`	an object of class `3daling` as obtained from function `pdbaln` or `read.fasta.pdb`; a xyz matrix containing the cartesian coordinates of C-alpha atoms.
`row.inds`	a numeric vector of indices pointing to the PDB structures to keep (rows in the `pdbs$ali` matrix).
`col.inds`	a numeric vector of indices pointing to the alignment columns to keep (columns in the `pdbs$ali` matrix).

Details

Utility function to remove structures, or trim off columns, in a 'pdbs' PDBs object.

Value

Returns an updated 'pdbs' PDBs object with the following components:

`xyz`	numeric matrix of aligned C-alpha coordinates.
`resno`	character matrix of aligned residue numbers.
`b`	numeric matrix of aligned B-factor values.
`chain`	character matrix of aligned chain identifiers.
`id`	character vector of PDB sequence/structure names.
`ali`	character matrix of aligned sequences.
`call`	the matched call.

Author(s)

Lars Skjaerven

References

Grant, B.J. et al. (2006) *Bioinformatics* **22**, 2695–2696.

See Also

pdbaln, gap.inspect, read.fasta,read.fasta.pdb,

Examples

```
## Not run:
## Fetch PDB files and split to chain A only PDB files
ids <- c("1a70_A", "1czp_A", "1frd_A", "1fxi_A", "1iue_A", "1pfd_A")
raw.files <- get.pdb(ids, path = "raw_pdbs")
files <- pdbsplit(raw.files, ids, path = "raw_pdbs/split_chain")

## Sequence Alignement, and connectivity check
pdbs <- pdbaln(files)

cons <- inspect.connectivity(pdbs)

## omit files with missing residues
pdbs.filter(pdbs, row.inds=which(cons))

## End(Not run)
```

---

pdbs2pdb *PDBs to PDB Converter*

Description

Convert a list of PDBs from an "pdbs" object to a list of pdb objects.

Usage

```
pdbs2pdb(pdbs, inds = NULL, rm.gaps = FALSE)
```

Arguments

pdbs	a list of class "pdbs" containing PDB file data, as obtained from read.fasta.pdb or pdbaln.
inds	a vector of indices that selects the PDB structures to convert.
rm.gaps	logical, if TRUE atoms in gap containing columns are removed in the output pdb objects.

Details

This function will generate a list of pdb objects from a "pdbs" class.

See examples for more details/

Value

Returns a list of pdb objects.

Author(s)

Lars Skjaerven

References

Grant, B.J. et al. (2006) *Bioinformatics* **22**, 2695–2696.

See Also

read.pdb, pdbaln, read.fasta.pdb.

Examples

```
## Not run:
## Fetch PDBs
pdb.ids <- c("1YX5_B", "3NOB", "1P3Q_U")
#outdir <- paste(tempdir(), "/raw_pdbs", sep="")
outdir = "raw_pdbs"
raw.files <- get.pdb(pdb.ids, path = outdir)

## Split PDBs by chain ID and multi-model records
all.files <- pdbsplit(raw.files, pdb.ids,
               path =paste(outdir, "/split_chain", sep=""))

## Align and fit
pdbs     <- pdbaln(all.files, fit=TRUE)

## Convert back to PDB objects
all.pdbs <- pdbs2pdb(pdbs)

## Access the first PDB object
## all.pdbs[[1]]

## Return PDB objects consisting of only
## atoms in non-gap positions
all.pdbs <- pdbs2pdb(pdbs, rm.gaps=TRUE)


## End(Not run)
```

pdbseq *Extract The Aminoacid Sequence From A PDB Object*

Description

Return a vector of the one-letter IUPAC or three-letter PDB style aminoacid codes from a given PDB object.

Usage

```
pdbseq(pdb, inds = NULL, aa1 = TRUE)
```

Arguments

pdb	a PDB structure object obtained from read.pdb.
inds	a list object of ATOM and XYZ indices as obtained from atom.select.
aa1	logical, if TRUE then the one-letter IUPAC sequence is returned. IF FALSE then the three-letter PDB style sequence is returned.

Details

See the functions atom.select and aa321 for further details.

Value

A character vector of aminoacid codes.

Author(s)

Barry Grant

References

Grant, B.J. et al. (2006) *Bioinformatics* **22**, 2695–2696.

For a description of IUPAC one-letter codes see:
http://www.chem.qmul.ac.uk/iupac/AminoAcid/

For a description of PDB residue codes see Appendix 4:
http://msdlocal.ebi.ac.uk/docs/pdb_format/appendix.html

See Also

read.pdb, atom.select, aa321, read.fasta

Examples

```
## Not run:
pdb <- read.pdb( "5p21" )
pdbseq(pdb)

## End(Not run)
```

---

pdbsplit *Split a PDB File Into Separate Files, One For Each Chain.*

Description

Split a Protein Data Bank (PDB) coordinate file into new separate files with one file for each chain.

Usage

```
pdbsplit(pdb.files, ids = NULL, path = "split_chain", overwrite=TRUE,
         verbose = FALSE, ncore = 1, ...)
```

Arguments

pdb.files	a character vector of PDB file names.
ids	a character vector of PDB and chain identifiers (of the form: 'pdbId_chainId', e.g. '1bg2_A'). Used for filtering chain IDs for output (in the above example only chain A would be produced).
path	output path for chain-split files.
overwrite	logical, if FALSE the PDB structures will not be read and written if split files already exist.
verbose	logical, if TRUE details of the PDB header and chain selections are printed.
ncore	number of CPU cores used for the calculation. ncore>1 requires package 'parallel' be installed.
...	additional arguments to read.pdb. Useful e.g. for parsing multi model PDB files, including ALT records etc. in the output files.

Details

This function will produce single chain PDB files from multi-chain input files. By default all separate filenames are returned. To return only a subset of select chains the optional input 'ids' can be provided to filter the output (e.g. to fetch only chain C, of a PDB object with additional chains A+B ignored). See examples section for further details.

Note that multi model atom records will only split into individual PDB files if multi=TRUE, else they are omitted. See examples.

Value

Returns a character vector of chain-split file names.

Author(s)

Barry Grant

References

Grant, B.J. et al. (2006) *Bioinformatics* **22**, 2695–2696.

For a description of PDB format (version3.3) see:
http://www.wwpdb.org/documentation/format33/v3.3.html.

See Also

read.pdb, atom.select, write.pdb, get.pdb.

Examples

```
## Not run:
  ## Save separate PDB files for each chain of a local or on-line file
  pdbsplit( get.pdb("2KIN", URLonly=TRUE) )


  ## Split several PDBs by chain ID and multi-model records
  raw.files <- get.pdb( c("1YX5", "3NOB") , URLonly=TRUE)
  chain.files <- pdbsplit(raw.files,  path=tempdir(), multi=TRUE)
  basename(chain.files)


  ## Output only desired pdbID_chainID combinations
  ## for the last entry (1f9j), fetch all chains
  ids <- c("1YX5_A", "3NOB_B", "1F9J")
  raw.files <- get.pdb( ids , URLonly=TRUE)
  chain.files <- pdbsplit(raw.files, ids, path=tempdir())
  basename(chain.files)

## End(Not run)
```

---

pfam *Download Pfam FASTA Sequence Alignment*

Description

Downloads FASTA sequence alignment from the Pfam database.

Usage

```
pfam(id, alignment = "seed", verbose = FALSE)
```

Arguments

id	the Pfam familiy identifier (e.g 'Piwi') or accession (e.g. 'PF02171').
alignment	the alignment type. Allowed values are: 'seed', 'ncbi', 'full', 'metagenomics'.
verbose	logical, if TRUE details of the download process is printed.

Details

This is a basic function to download a multiple sequence alignment for a protein family from the Pfam database.

Value

A 'fasta' object with the following components:

ali	an alignment character matrix with a row per sequence and a column per equivalent aminoacid/nucleotide.
ids	sequence names as identifiers.
call	the matched call.

Note

Full more information on the Pfam database:
http://pfam.xfam.org

Author(s)

Lars Skjaerven

References

Grant, B.J. et al. (2006) *Bioinformatics* **22**, 2695–2696.

See Also

read.fasta, hmmer, get.seq, uniprot

Examples

```
aln <- pfam("piwi")
aln <- pfam("PF02171")

seq <- get.seq("1rx2_A", outfile = tempfile())
hmm <- hmmer(seq, type="hmmscan", db="pfam")
aln <- pfam(hmm$acc[1])
```

plot.bio3d *Plots with marginal SSE annotation*

Description

Draw a standard scatter plot with optional secondary structure in the marginal regions.

Usage

```
## S3 method for class 'bio3d'
plot(x, y = NULL, type = "h", main = "", sub = "", xlim = NULL, ylim = NULL,
                  ylim2zero = TRUE, xlab = NULL, ylab = NULL, axes = TRUE,
              ann = par("ann"), col = par("col"), sse = NULL, top = TRUE, bot = TRUE,
              helix.col = "gray20", sheet.col = "gray80", sse.border = FALSE, ...)
```

Arguments

x	the x coordinates for the plot. Any reasonable way of defining the coordinates is acceptable. See the function 'xy.coords' for details.
y	the y coordinates for the plot, see above.
type	one-character string giving the type of plot desired. The following values are possible, (for details, see 'plot'): 'p' for points, 'l' for lines, 'o' for overplotted points and lines, 'b', 'c') for points joined by lines, 's' and 'S' for stair steps and 'h' for histogram-like vertical lines. Finally, 'n' does not produce any points or lines.
main	a main title for the plot, see also 'title'.
sub	a sub-title for the plot.
xlim	the x limits (x1,x2) of the plot. Note that x1 > x2 is allowed and leads to a reversed axis.
ylim	the y limits of the plot.
ylim2zero	logical, if TRUE the y-limits are forced to start at zero.
xlab	a label for the x axis, defaults to a description of 'x'.
ylab	a label for the y axis, defaults to a description of 'y'.
axes	a logical value indicating whether both axes should be drawn on the plot. Use graphical parameter 'xaxt' or 'yaxt' to suppress just one of the axes.
ann	a logical value indicating whether the default annotation (title and x and y axis labels) should appear on the plot.
col	The colors for lines and points. Multiple colours can be specified so that each point is given its own color. If there are fewer colors than points they are recycled in the standard fashion. Lines are plotted in the first colour specified.
sse	secondary structure object as returned from dssp or stride.
top	logical, if TRUE rectangles for each sse are drawn towards the top of the plotting region.

bot	logical, if TRUE rectangles for each sse are drawn towards the bottom of the plotting region.
helix.col	The colors for rectangles representing alpha helices.
sheet.col	The colors for rectangles representing beta strands.
sse.border	The border color for all sse rectangles.
...	other graphical parameters.

Details

See the functions 'plot.default', dssp and stride for further details.

Value

Called for its effect.

Note

Be sure to check the correspondence of your 'sse' object with the 'x' values being plotted as no internal checks are performed.

Author(s)

Barry Grant

References

Grant, B.J. et al. (2006) *Bioinformatics* **22**, 2695–2696.

See Also

plot.default, dssp, stride

Examples

```
## Plot of B-factor values along with secondary structure from PDB
pdb <- read.pdb( "1bg2" )
plot.bio3d(pdb$atom[pdb$calpha,"b"], sse=pdb, ylab="B-factor")


## Not run:
## Calculate secondary structure
#sse <- stride(pdb, resno=FALSE)
sse <- dssp(pdb, resno=FALSE)

## Plot of B-factor values along with calculated secondary structure
plot.bio3d(pdb$atom[pdb$calpha,"b"], sse=sse, ylab="B-factor", typ="l",
col="blue", lwd=2)

## End(Not run)
```

plot.blast *Plot a Summary of BLAST Hit Statistics.*

Description

Produces a number of basic plots that should facilitate hit selection from the match statistics of a BLAST result.

Usage

```
 ## S3 method for class 'blast'
plot(x, cutoff = NULL, cut.seed=NULL, cluster=TRUE, mar=c(2, 5, 1, 1), cex=1.5, ...)
```

Arguments

x	BLAST results as obtained from the function blast.pdb.
cutoff	A numeric cutoff value, in terms of minus the log of the evalue, for returned hits. If null then the function will try to find a suitable cutoff near 'cut.seed' which can be used as an initial guide (see below).
cut.seed	A numeric seed cutoff value, used for initial cutoff estimation. If null then a seed position is set to the point of largest drop-off in normalized scores (i.e. the biggest jump in E-values).
cluster	Logical, if TRUE (and 'cutoff' is null) a clustering of normalized scores is performed to partition hits in groups by similarity to query. If FALSE the partition point is set to the point of largest drop-off in normalized scores.
mar	A numerical vector of the form c(bottom, left, top, right) which gives the number of lines of margin to be specified on the four sides of the plot.
cex	a numerical single element vector giving the amount by which plot labels should be magnified relative to the default.
...	extra plotting arguments.

Details

Examining plots of BLAST alignment lengths, scores, E-values and normalized scores (-log(E-Value), see 'blast.pdb' function) can aid in the identification sensible hit similarity thresholds.

If a 'cutoff' value is not supplied then a basic hierarchical clustering of normalized scores is performed with initial group partitioning implemented at a hopefully sensible point in the vicinity of 'h=cut.seed'. Inspection of the resultant plot can then be use to refine the value of 'cut.seed' or indeed 'cutoff'. As the 'cutoff' value can vary depending on the desired application and indeed the properties of the system under study it is envisaged that 'plot.blast' will be called multiple times to aid selection of a suitable 'cutoff' value. See the examples below for further details.

Value

Produces a plot on the active graphics device and returns a three component list object:

`hits`	an ordered matrix detailing the subset of hits with a normalized score above the chosen cutoff. Database identifiers are listed along with their cluster group number.
`pdb.id`	a character vector containing the PDB database identifier of each hit above the chosen threshold.
`gi.id`	a character vector containing the gi database identifier of each hit above the chosen threshold.

Note

TO BE IMPROVED.

Author(s)

Barry Grant

References

Grant, B.J. et al. (2006) *Bioinformatics* **22**, 2695–2696.

See Also

`blast.pdb`

Examples

```
b2 <-  blast.pdb( pdbseq(read.pdb( get.pdb("4q21", URLonly=TRUE) )) )
raw.hits <- plot.blast(b2)
top.hits <- plot.blast(b2, 188)
head(top.hits$hits)


## Not run:
blast <- blast.pdb( pdbseq(read.pdb( get.pdb("2BN3", URLonly=TRUE) )))
raw.hits <- plot(blast)
top.hits <- plot(blast, cut.seed=20)

head(top.hits$pdb.id)
#pdbFiles <- get.pdb(substr(top.hits$pdb.id, 1, 4), path="downloadedPDBs")
#pdbsplit(pdbFiles, path="downloadedPDBs/PDB_chains")

## End(Not run)
```

plot.cna *Protein Structure Network Plots in 2D and 3D.*

Description

Plot a protein dynamic network as obtained from the *cna* function.

Usage

```
  ## S3 method for class 'cna'
plot(x, pdb = NULL, weights=NULL, vertex.size=NULL,
         layout=NULL, col=NULL, full=FALSE, scale=TRUE, color.edge = FALSE, ...)
```

Arguments

x	A protein network graph object as obtained from the 'cna' function.
pdb	A PDB structure object obtained from 'read.pdb'. If supplied this will be used to guide the network plot 'layout', see 'layout.cna' for details.
weights	A numeric vector containing the edge weights for the network.
vertex.size	A numeric vector of node/community sizes. If NULL the size will be taken from the input network graph object 'x'. Typically for 'full=TRUE' nodes will be of an equal size and for 'full=FALSE' community node size will be proportional to the residue membership of each community.
layout	Either a function or a numeric matrix. It specifies how the vertices will be placed on the plot. See 'layout.cna'.
col	A vector of colors used for node/vertex rendering. If NULL these values are taken from the input network 'V(x$community.network)$color'.
full	Logical, if TRUE the full all-atom network rather than the clustered community network will be plotted.
scale	Logical, if TRUE weights are scaled with respect to the network.
color.edge	Logical, if TRUE edges are colored with respect to their weights.
...	Additional graphical parameters for 'plot.igraph'.

Details

This function calls 'plot.igraph' from the igraph package to plot cna networks the way we like them.

The plot layout is user settable, we like the options of: 'layout.cna', 'layout.fruchterman.reingold', 'layout.mds' or 'layout.svd'. Note that first of these uses PDB structure information to produce a more meaningful layout.

Extensive plot modifications are possible by setting additional graphical parameters (...). These options are detailed in 'igraph.plotting'. Common parameters to alter include:

vertex.label: Node labels, `V(x$network)$name`. Use NA to omit.

vertex.label.color: Node label colors, see also `vertex.label.cex` etc.

edge.color: Edge colors, `E(x$network)$color`.

mark.groups: Community highlighting, a community list object, see also `mark.col` etc.

Value

Produces a network plot on the active graphics device. Also returns the plot layout coordinates silently, which can be passed to the 'identify.cna' function.

Note

Be sure to check the correspondence of your 'pdb' object with your network object 'x', as few internal checks are currently performed by the 'layout.cna' function.

Author(s)

Barry Grant and Guido Scarabelli

References

Grant, B.J. et al. (2006) *Bioinformatics* **22**, 2695–2696.

See Also

plot.igraph, plot.communities, igraph.plotting

Examples

```
require(igraph)

##-- Build a CNA object
pdb <- read.pdb("4Q21")
modes <- nma(pdb)
cij <- dccm(modes)
net <- cna(cij, cutoff.cij=0.2)

# Plot coarse grain network based on dynamically coupled communities
xy <- plot.cna(net)
#plot.dccm2(cij, margin.segments=net$communities$membership)

# Chose a different PDB informed layout for plot
plot.cna(net, pdb)

# Play with plot layout and colors...
plot.cna(net, layout=layout.mds(net$community.network), col=c("blue","green") )


## Not run:
# Plot full residue network colored by communities - will be slow due to number of edges!!
plot.cna(net, pdb, full=TRUE)

# Alter plot settings
plot.cna(net, pdb, full=TRUE, vertex.size=3, weights=1, vertex.label=NA)

## End(Not run)
```

plot.core *Plot Core Fitting Progress*

Description

Plots the total ellipsoid volume of core positions versus core size at each iteration of the core finding process.

Usage

```
## S3 method for class 'core'
plot(x, y = NULL, type = "h", main = "", sub = "",
   xlim = NULL, ylim = NULL, xlab = "Core Size (Number of Residues)",
   ylab = "Total Ellipsoid Volume (Angstrom^3)", axes = TRUE,
   ann = par("ann"), col = par("col"), ...)
```

Arguments

x	a list object obtained with the function core.find from which the 'volume' component is taken as the x coordinates for the plot.
y	the y coordinates for the plot.
type	one-character string giving the type of plot desired.
main	a main title for the plot, see also 'title'.
sub	a sub-title for the plot.
xlim	the x limits of the plot.
ylim	the y limits of the plot.
xlab	a label for the x axis.
ylab	a label for the y axis.
axes	a logical value indicating whether both axes should be drawn.
ann	a logical value indicating whether the default annotation (title and x and y axis labels) should appear on the plot.
col	The colors for lines and points. Multiple colours can be specified so that each point is given its own color. If there are fewer colors than points they are recycled in the standard fashion.
...	extra plotting arguments.

Value

Called for its effect.

Note

The produced plot can be useful for deciding on the core/non-core boundary.

Author(s)

Barry Grant

References

Grant, B.J. et al. (2006) *Bioinformatics* **22**, 2695–2696.

See Also

core.find, print.core

Examples

```
## Not run:

##-- Generate a small kinesin alignment and read corresponding structures
pdbfiles <- get.pdb(c("1bg2","2ncd","1i6i","1i5s"), URLonly=TRUE)
pdbs <- pdbaln(pdbfiles)

##-- Find 'core' positions
core <- core.find(pdbs)
plot(core)

##-- Fit on these relatively invarient subset of positions
core.inds <- print(core)
xyz <- pdbfit(pdbs, core.inds, outpath="corefit_structures")

##-- Compare to fitting on all equivalent positions
xyz2 <- pdbfit(pdbs)

## Note that overall RMSD will be higher but RMSF will
##  be lower in core regions, which may equate to a
##  'better fit' for certain applications
gaps <- gap.inspect(pdbs$xyz)
rmsd(xyz[,gaps$f.inds])
rmsd(xyz2[,gaps$f.inds])

plot(rmsf(xyz[,gaps$f.inds]), typ="l", col="blue", ylim=c(0,9))
points(rmsf(xyz2[,gaps$f.inds]), typ="l", col="red")


## End(Not run)
```

---

plot.dccm *DCCM Plot*

Description

Plot a dynamical cross-correlation matrix.

Usage

```
## S3 method for class 'dccm'
plot(x, sse=NULL, colorkey=TRUE,
                    at=c(-1, -0.75, -0.5,  -0.25, 0.25, 0.5, 0.75, 1),
                    main="Residue Cross Correlation",
                    helix.col = "gray20", sheet.col = "gray80",
                    inner.box=TRUE, outer.box=FALSE,
                    xlab="Residue No.", ylab="Residue No.",
                  margin.segments=NULL, segment.col=vmd.colors(), segment.min=1, ...)
```

Arguments

x	a numeric matrix of atom-wise cross-correlations as output by the 'dccm' function.
sse	secondary structure object as returned from dssp, stride or read.pdb.
colorkey	logical, if TRUE a key is plotted.
at	numeric vector specifying the levels to be colored.
main	a main title for the plot.
helix.col	The colors for rectangles representing alpha helices.
sheet.col	The colors for rectangles representing beta strands.
inner.box	logical, if TRUE an outer box is drawn.
outer.box	logical, if TRUE an outer box is drawn.
xlab	a label for the x axis.
ylab	a label for the y axis.
margin.segments	a numeric vector of cluster membership as obtained from cutree() or other community detection method. This will be used for bottom and left margin annotation.
segment.col	a vector of colors used for each cluster group in margin.segments.
segment.min	a single element numeric vector that will cause margin.segments with a length below this value to be excluded from the plot.
...	additional graphical parameters for contourplot.

Details

See the 'contourplot' function from the lattice package for plot customization options, and the functions dssp and stride for further details.

Value

Called for its effect.

Note

Be sure to check the correspondence of your 'sse' object with the 'cij' values being plotted as no internal checks are currently performed.

Author(s)

Barry Grant

References

Grant, B.J. et al. (2006) *Bioinformatics* **22**, 2695–2696.

See Also

plot.bio3d, plot.dmat, filled.contour, contour, image plot.default, dssp, stride

Examples

```
## Not run:
  ##-- Read example trajectory file
  trtfile <- system.file("examples/hivp.dcd", package="bio3d")
  trj <- read.dcd(trtfile)

  ## Read the starting PDB file to determine atom correspondence
  pdbfile <- system.file("examples/hivp.pdb", package="bio3d")
  pdb <- read.pdb(pdbfile)

  ## select residues 24 to 27 and 85 to 90 in both chains
  inds <- atom.select(pdb,"///24:27,85:90///CA/")

  ## lsq fit of trj on pdb
  xyz <- fit.xyz(pdb$xyz, trj, fixed.inds=inds$xyz, mobile.inds=inds$xyz)

  ## Dynamic cross-correlations of atomic displacements
  cij <- dccm(xyz)

  ## Default plot
  plot.dccm(cij)

  ## Change the color scheme and the range of colored data levels
  plot.dccm(cij, contour=F, col.regions=bwr.colors(200), at=seq(-1,1,by=0.01) )

  ## Add secondary structure annotation to plot margins
  sse <- dssp(read.pdb("1W5Y"), resno=FALSE)
  plot.dccm(cij, sse=sse)

  ## Add additional margin annotation for chains..
  ch <- ifelse(pdb$atom[pdb$calpha,"chain"]=="A", 1,2)
  plot.dccm(cij, sse=sse, margin.segments=ch)

  ## Plot with cluster annotation from dynamic network analysis
  #net <- cna(cij)
  #plot.dccm2(cij, margin.segments=net$raw.communities$membership)

  ## Focus on major communities (i.e. exclude those below a certain total length)
  #plot.dccm2(cij, margin.segments=net$raw.communities$membership, segment.min=25)
```

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

`plot.dmat` *Plot Distance Matrix*

Description

Plot a distance matrix (DM) or a difference distance matrix (DDM).

Usage

```
## S3 method for class 'dmat'
plot(x, key = TRUE, resnum.1 = c(1:ncol(x)), resnum.2 = resnum.1,
          axis.tick.space = 20, zlim = range(x, finite = TRUE),
          nlevels = 20, levels = pretty(zlim, nlevels),
          color.palette = bwr.colors,
          col = color.palette(length(levels) - 1),
          axes = TRUE, key.axes, xaxs = "i", yaxs = "i", las = 1,
          grid = TRUE, grid.col = "yellow", grid.nx = floor(ncol(x)/30),
          grid.ny = grid.nx, center.zero = TRUE, flip=TRUE, ...)
```

Arguments

`x`	a numeric distance matrix generated by the function `dm`.
`key`	logical, if TRUE a color key is plotted.
`resnum.1`	a vector of residue numbers for annotating the x axis.
`resnum.2`	a vector of residue numbers for annotating the y axis.
`axis.tick.space`	the separation between each axis tick mark.
`zlim`	z limits for the distances to be plotted.
`nlevels`	if `levels` is not specified, the range of 'z' values is divided into approximately this many levels.
`levels`	a set of levels used to partition the range of 'z'. Must be *strictly* increasing (and finite). Areas with 'z' values between consecutive levels are painted with the same color.
`color.palette`	a color palette function, used to assign colors in the plot.
`col`	an explicit set of colors to be used in the plot. This argument overrides any palette function specification.
`axes`	logical, if TRUE plot axes are drawn.
`key.axes`	statements which draw axes on the plot key. It overrides the default axis.
`xaxs`	the x axis style. The default is to use internal labeling.

yaxs	the y axis style. The default is to use internal labeling.
las	the style of labeling to be used. The default is to use horizontal labeling.
grid	logical, if TRUE overlaid grid is drawn.
grid.col	color of the overlaid grid.
grid.nx	number of grid cells in the x direction.
grid.ny	number of grid cells in the y direction.
center.zero	logical, if TRUE levels are forced to be equidistant around zero, assuming that zlim ranges from less than to more than zero.
flip	logical, indicating whether the second axis should be fliped.
...	additional graphical parameters for image.

Value

Called for its effect.

Note

This function is based on the `layout` and legend key code in the function `filled.contour` by Ross Ihaka. As with `filled.contour` the output is a combination of two plots: the legend and (in this case) `image` (rather than a contour plot).

Author(s)

Barry Grant

References

Grant, B.J. et al. (2006) *Bioinformatics* **22**, 2695–2696.T

Much of this function is based on the `filled.contour` function by Ross Ihaka.

See Also

dm, filled.contour, contour, image

Examples

```
# Read PDB file
pdb <- read.pdb( "1bg2" )

# DM
d <- dm(pdb,"calpha")


## Not run:
# Plot DM
##filled.contour(d, nlevels = 4)
##plot(d)
```

```
plot(d,
     resnum.1 = pdb$atom[pdb$calpha,"resno"],
     color.palette = mono.colors,
     xlab="Residue Number", ylab="Residue Number")


# Downlaod and align two PDB files
pdbs <- pdbaln( get.pdb( c( "4q21", "521p"), path=tempdir(), overwrite=TRUE))

# Get distance matrix
a <- dm(pdbs$xyz[1,])
b <- dm(pdbs$xyz[2,])

# Calculate DDM
c <- a - b

# Plot DDM
plot(c,key=FALSE, grid=FALSE)

plot(c, axis.tick.space=10,
     resnum.1=pdbs$resno[1,],
     resnum.2=pdbs$resno[2,],
     grid.col="black",
     xlab="Residue No. (4q21)", ylab="Residue No. (521p)")


## End(Not run)
```

---

plot.enma *Plot eNMA Results*

Description

Produces a plot of atomic fluctuations of aligned normal modes.

Usage

```
## S3 method for class 'enma'
plot(x,
        pdbs = NULL, conservation = NULL, variance = FALSE,
        spread = FALSE, offset = 1,
        col = NULL, signif = FALSE,
        pcut = 0.005, qcut = 0.04,
        xlab = "Alignment Position",
        ylab=c("Fluctuations", "Fluct.variance", "Seq.conservation"),
        xlim = NULL, ylim = NULL,
        mar = c(4, 5, 2, 2), ...)
```

Arguments

x	the results of ensemble NMA obtained with nma.pdbs. Alternatively, a matrix in the similar format as enma$fluctuations can be provided.
pdbs	an object of class 'pdbs' in which the 'enma' object x was obtained from. If provided SSE data of the first structure of pdbs will drawn.
conservation	logical, if TRUE sequence conservation is plotted. Alternatively, provide the conservation assement method ('similarity', 'identity', 'entropy22', or 'entropy10'). A numeric vector of residue conservation values are also allowed.
variance	logical, if TRUE fluctuation variance is plotted.
spread	logical, if TRUE the fluctuation profiles are spread - i.e. not on top of each other.
offset	numerical offset value in use when 'spread=TRUE'.
col	a character vector of plotting colors. Used also to group fluctuation profiles when 'spread=TRUE'. NA values in col will omit the corresponding fluctuation profile in the plot.
signif	logical, if TRUE significance of difference is plotted.
pcut	P-value cutoff for the significance.
qcut	Cutoff for the minimal difference of mean fluctuation to plot the significance.
xlab	a label for the x axis.
ylab	labels for the y axes.
mar	a numerical vector of the form c(bottom, left, top, right) which gives the number of lines of margin to be specified on the four sides of the plot.
xlim	the x limits of the plot.
ylim	the y limits of the plot.
...	extra plotting arguments passed to plot.bio3d that effect the atomic fluctuations plot only.

Details

plot.enma produces a fluctuation plot of aligned nma objects.

Value

Called for its effect.

Author(s)

Lars Skjaerven, Barry Grant

References

Grant, B.J. et al. (2006) *Bioinformatics* **22**, 2695–2696.

See Also

nma.pdbs, nma, plot.bio3d, entropy.

Examples

```
## Not run:
ids <- c("1a70_A", "1czp_A", "1frd_A", "1fxi_A", "1iue_A", "1pfd_A")
raw.files <- get.pdb(ids, path = "raw_pdbs")
files <- pdbsplit(raw.files, ids, path = "raw_pdbs/split_chain")

## Sequence Alignement
pdbs <- pdbaln(files)

## Normal mode analysis on aligned data
all.modes <- nma.pdbs(pdbs, rm.gaps=TRUE)

## Plot fluctuations
plot.enma(all.modes, pdbs=pdbs, conservation=TRUE)

## group and spread fluctuation profiles
grps <- rep(NA, length(pdbs$id))
grps[c(2,3)]=1
grps[c(4,5)]=2

plot.enma(all.modes, pdbs=pdbs, col=grps, spread=TRUE)

## End(Not run)
```

`plot.fasta` *Plot a Multiple Sequence Alignment*

Description

Produces a schematic representation of a multiple sequence alignment.

Usage

```
## S3 method for class 'fasta'
plot(x, plot.labels = TRUE,
                     plot.bars = TRUE,
                     plot.lines = FALSE,
                     plot.axis = TRUE,
                     seq.index = NULL,
                     color.conserved = FALSE,
                     cutoff=0.5,
                     col=NULL,
                     bars.scale=2,
                     row.spacing=0.5,
                     aln.col="grey50",
                     cex.text=1, add=FALSE, ...)
```

Arguments

x	multiple sequence alignement of class 'fasta' as obtained from seqaln.
plot.labels	logical, if TRUE labels will be printed next to the sequence bar.
plot.bars	logical, if TRUE an additional bar representing sequence conservation will be plotted.
plot.lines	logical, if TRUE sequence conservation will be represented with a plot.
plot.axis	logical, if TRUE x-axis will be plotted.
seq.index	printed tick labels will correspond to the sequence of the provided index.
color.conserved	logical, if TRUE conserved residues will be colored according to "clustal" coloring scheme.
cutoff	conservation 'cutoff' value below which alignment columns are not colored.
col	character vector with color codes for the conservation bars. By default, heat.colors will be used.
bars.scale	scaling factor for the height of the conservation bar when 'plot.bars=TRUE'.
row.spacing	space between the sequence bars.
aln.col	color of the alignment bars.
cex.text	scaling factor for the labels.
add	logical, if TRUE plot.new() will not be called.
...	additional arguments not paseed anywhere.

Details

plot.fasta is a utility function for producting a schematic representation of a multiple sequence alignment.

Value

Called for its effect.

Author(s)

Lars Skjaerven

References

Grant, B.J. et al. (2006) *Bioinformatics* **22**, 2695–2696.

See Also

seqaln, read.fasta, entropy, aln2html.

Examples

```
# Read alignment
aln<-read.fasta(system.file("examples/kif1a.fa",package="bio3d"))

## alignment plot
plot(aln)

## Not run:
infile <- "http://pfam.sanger.ac.uk/family/PF00071/alignment/seed/format?format=fasta"
aln <- read.fasta( infile )
plot(aln)

## End(Not run)
```

`plot.hmmer` *Plot a Summary of HMMER Hit Statistics.*

Description

Produces a number of basic plots that should facilitate hit selection from the match statistics of a HMMER result.

Usage

```
 ## S3 method for class 'hmmer'
plot(x, cutoff = NULL, cut.seed=NULL, cluster=TRUE, mar=c(2, 5, 1, 1), cex=1.1, ...)
```

Arguments

`x`	HMMER results as obtained from the function `hmmer`.
`cutoff`	A numeric cutoff value, in terms of minus the log of the evalue, for returned hits. If null then the function will try to find a suitable cutoff near 'cut.seed' which can be used as an initial guide (see below).
`cut.seed`	A numeric seed cutoff value, used for initial cutoff estimation. If null then a seed position is set to the point of largest drop-off in normalized scores (i.e. the biggest jump in E-values).
`cluster`	Logical, if TRUE (and 'cutoff' is null) a clustering of normalized scores is performed to partition hits in groups by similarity to query. If FALSE the partition point is set to the point of largest drop-off in normalized scores.
`mar`	A numerical vector of the form c(bottom, left, top, right) which gives the number of lines of margin to be specified on the four sides of the plot.
`cex`	a numerical single element vector giving the amount by which plot labels should be magnified relative to the default.
`...`	extra plotting arguments.

Details

Examining plots of HMMER scores, E-values and normalized scores (-log(E-Value), see 'hmmer' function) can aid in the identification sensible hit similarity thresholds.

If a 'cutoff' value is not supplied then a basic hierarchical clustering of normalized scores is performed with initial group partitioning implemented at a hopefully sensible point in the vicinity of 'h=cut.seed'. Inspection of the resultant plot can then be use to refine the value of 'cut.seed' or indeed 'cutoff'. As the 'cutoff' value can vary depending on the desired application and indeed the properties of the system under study it is envisaged that 'plot.hmmer' will be called multiple times to aid selection of a suitable 'cutoff' value. See the examples below for further details.

Value

Produces a plot on the active graphics device and returns a three component list object:

`hits`	an ordered matrix detailing the subset of hits with a normalized score above the chosen cutoff. Database identifiers are listed along with their cluster group number.
`acc`	a character vector containing the database accession identifier of each hit above the chosen threshold.
`inds`	a numeric vector containing the indices of the hits relative to the input hmmer object.

Note

TO BE IMPROVED.

Author(s)

Barry Grant, Lars Skjaerven

References

Grant, B.J. et al. (2006) *Bioinformatics* **22**, 2695–2696.

See Also

`hmmer`, `blast.pdb`

Examples

```
##- PHMMER
seq <- get.seq("2abl_A", outfile = tempfile())
res <- hmmer(seq, db="pdb")
plot.hmmer(res)
```

`plot.nma` *Plot NMA Results*

Description

Produces eigenvalue/frequency spectrum plots and an atomic fluctuations plot.

Usage

```
## S3 method for class 'nma'
plot(x, pch = 16, col = par("col"), cex=0.8, mar=c(6, 4, 2, 2),...)
```

Arguments

`x`	the results of normal modes analysis obtained with `nma`.
`pch`	a vector of plotting characters or symbols: see 'points'.
`col`	a character vector of plotting colors.
`cex`	a numerical single element vector giving the amount by which plotting text and symbols should be magnified relative to the default.
`mar`	A numerical vector of the form c(bottom, left, top, right) which gives the number of lines of margin to be specified on the four sides of the plot.
`...`	extra plotting arguments passed to `plot.bio3d` that effect the atomic fluctuations plot only.

Details

`plot.nma` produces an eigenvalue (or frequency) spectrum plot together with a plot of the atomic fluctuations.

Value

Called for its effect.

Author(s)

Lars Skjaerven

References

Grant, B.J. et al. (2006) *Bioinformatics* **22**, 2695–2696.

See Also

`nma`, `plot.bio3d`

Examples

```
## Fetch structure
pdb <- read.pdb( system.file("examples/1hel.pdb", package="bio3d") )

## Calculate modes
modes <- nma(pdb)

plot(modes, sse=pdb)
```

---

plot.pca *Plot PCA Results*

Description

Produces a z-score plot (conformer plot) and an eigen spectrum plot (scree plot).

Usage

```
## S3 method for class 'pca'
plot(x, pch = 16, col = par("col"), cex=0.8, mar=c(4, 4, 1, 1),...)
## S3 method for class 'pca.scree'
plot(x, y = NULL, type = "o", pch = 18,
         main = "", sub = "", xlim = c(0, 20), ylim = NULL,
         ylab = "Proporton of Variance (%)",
         xlab = "Eigenvalue Rank", axes = TRUE, ann = par("ann"),
         col = par("col"), lab = TRUE, ...)
## S3 method for class 'pca.score'
plot(x, inds=NULL, col=rainbow(nrow(x)), lab = "", ...)
```

Arguments

x	the results of principal component analysis obtained with pca.xyz.
pch	a vector of plotting characters or symbols: see 'points'.
col	a character vector of plotting colors.
cex	a numerical single element vector giving the amount by which plotting text and symbols should be magnified relative to the default.
mar	A numerical vector of the form c(bottom, left, top, right) which gives the number of lines of margin to be specified on the four sides of the plot.
inds	row indices of the conformers to label.
lab	a character vector of plot labels.
y	the y coordinates for the scree plot.
type	one-character string giving the type of plot desired.

main	a main title for the plot, see also 'title'.
sub	a sub-title for the plot.
xlim	the x limits of the plot.
ylim	the y limits of the plot.
ylab	a label for the y axis.
xlab	a label for the x axis.
axes	a logical value indicating whether both axes should be drawn.
ann	a logical value indicating whether the default annotation (title and x and y axis labels) should appear on the plot.
...	extra plotting arguments.

Details

plot.pca is a wrapper calling both plot.pca.score and plot.pca.scree resulting in a 2x2 plot with three score plots and one scree plot.

Value

Called for its effect.

Author(s)

Barry Grant

References

Grant, B.J. et al. (2006) *Bioinformatics* **22**, 2695–2696.

See Also

pca.xyz, plot.bio3d

Examples

```
data(transducin)
attach(transducin, warn.conflicts=FALSE)

pc.xray <- pca.xyz(pdbs$xyz[, gap.inspect(pdbs$xyz)$f.inds])
plot(pc.xray)

## color by nucleotide state
vcolors <- annotation[, "color"]
plot(pc.xray, col=vcolors)

## add labels
#labs <- rownames(annotation)
#inds <- c(2,7)
#plot.pca.score(pc.xray, inds=inds, col=vcolors, lab=labs)
```

```
## color by seq identity
#ide <- seqidentity(pdbs$ali)
#hc <- hclust(as.dist(1-ide))
#grps <- cutree(hc, h=0.2)
#vcolors <- rainbow(max(grps))[grps]
#plot(pc.xray, inds=inds, col=vcolors, lab=labs)

detach(transducin)
```

`plot.pca.loadings` *Plot Residue Loadings along PC1 to PC3*

Description

Plot residue loadings along PC1 to PC3 from a given xyz C-alpha matrix of `loadings`.

Usage

```
## S3 method for class 'pca.loadings'
plot(x, resnums = seq(1, (length(x[, 1])/3), 25), ...)
```

Arguments

`x`	the results of principal component analysis obtained from `pca.xyz`, or just the loadings returned from `pca.xyz`.
`resnums`	a numeric vector of residue numbers.
`...`	extra plotting arguments.

Value

Called for its effect.

Author(s)

Barry Grant

References

Grant, B.J. et al. (2006) *Bioinformatics* **22**, 2695–2696.

See Also

`pca.xyz`, `plot.pca`

Examples

```
data(transducin)
attach(transducin, warn.conflicts=FALSE)
pc.xray <- pca.xyz(pdbs$xyz[, gap.inspect(pdbs$xyz)$f.inds])
plot.pca.loadings(pc.xray$U)

detach(transducin)
```

`plot.rmsip` *Plot RMSIP Results*

Description

Produces a heat plot of RMSIP (Root mean square inner product) for the visualization of modes similarity.

Usage

```
## S3 method for class 'rmsip'
plot(x, xlab = NULL, ylab = NULL, col = gray(50:0/50),
      zlim=c(0,1), ...)
```

Arguments

`x`	an object of class `rmsip`.
`xlab`	a label for the x axis, defaults to 'a'.
`ylab`	a label for the y axis, defaults to 'b'.
`col`	a vector of colors for the RMSIP map (or overlap values).
`zlim`	the minimum and maximum 'z' values for which colors should be plotted.
`...`	additional arguments to function `image`.

Details

`plot.rmsip` produces a color image with the function `image`.

Value

Called for its effect.

Author(s)

Lars Skjaerven

References

Grant, B.J. et al. (2006) *Bioinformatics* **22**, 2695–2696.

See Also

rmsip, overlap, nma, image.

Examples

```
## Read PDB structure
pdb <- read.pdb( system.file("examples/1hel.pdb", package="bio3d") )

## Perform NMA
modes.a <- nma(pdb, ff="calpha")
modes.b <- nma(pdb, ff="anm")

## Calculate and plot RMSIP
r <- rmsip(modes.a, modes.b)
plot(r)
```

---

print.cna *Summarize and Print Features of a cna Network Graph*

Description

These functions attempt to summarize and print a cna network graph to the terminal in a human readable form.

Usage

```
  ## S3 method for class 'cna'
print(x, ...)
  ## S3 method for class 'cna'
summary(object, verbose=TRUE, ...)
```

Arguments

x	A cna network and community object as obtained from the function 'cna'.
object	A cna network and community object as obtained from the function 'cna'.
verbose	Logical, if TRUE a community summary table is prited to screen.
...	Extra arguments passed to the 'write.table' function.

Details

Simple summary and print methods for protein dynamic networks.

Value

The function summary.cna returns a list with the following components:

id	A community number/identifier vector with an element for each community.
size	A numeric community size vector, with elements giving the number of nodes within each community.
members	A lst detailing the nodes within each community.

Author(s)

Guido Scarabelli and Barry Grant

References

Grant, B.J. et al. (2006) *Bioinformatics* **22**, 2695–2696.

See Also

cna, print.igraph, str.igraph, igraph.plotting

Examples

```
## Load the correlation network
attach(hivp)

## Read the starting PDB file to determine atom correspondence
pdbfile <- system.file("examples/hivp.pdb", package="bio3d")
pdb <- read.pdb(pdbfile)

## Examine network composition
print(net)
x<- summary(net)
x$members[[2]]
```

---

print.core *Printing Core Positions and Returning Indices*

Description

Print method for core.find objects.

Usage

```
## S3 method for class 'core'
print(x, vol = NULL, ...)
```

Arguments

x	a list object obtained with the function core.find.
vol	the maximal cumulative volume value at which core positions are detailed.
...	additional arguments to 'print'.

Value

Returns a three component list of indices:

atom	atom indices of core positions
xyz	xyz indices of core positions
resno	residue numbers of core positions

Note

The produced plot.core function can be useful for deciding on the core/non-core boundary.

Author(s)

Barry Grant

References

Grant, B.J. et al. (2006) *Bioinformatics* **22**, 2695–2696.

See Also

core.find, plot.core

Examples

```
## Not run:
##-- Generate a small kinesin alignment and read corresponding structures
pdbfiles <- get.pdb(c("1bg2","2ncd","1i6i","1i5s"), URLonly=TRUE)
pdbs <- pdbaln(pdbfiles)

##-- Find 'core' positions
core <- core.find(pdbs)
plot(core)

##-- Fit on these relatively invarient subset of positions
core.inds <- print(core, vol=0.5)

print(core, vol=0.7)
print(core, vol=1.0)


## End(Not run)
```

print.fasta *Printing Sequence Alignments*

Description

Print method for fasta and pdbs sequence alignment objects.

Usage

```
## S3 method for class 'fasta'
print(x, alignment=TRUE, ...)
.print.fasta.ali(x, width = NULL, col.inds = NULL, numbers = TRUE, ...)
```

Arguments

x	a sequence alignment object as obtained from the functions read.fasta, read.fasta.pdb, pdbaln, seqaln, etc.
alignment	logical, if TRUE the sequence alignment will be printed to screen.
width	a single numeric value giving the number of residues per printed sequence block. By default this is determined from considering alignment identifier widths given a standard 85 column terminal window.
col.inds	an optional numeric vector that can be used to select subsets of alignment positions/columns for printing.
numbers	logical, if TRUE position numbers and a tick-mark every 10 positions are printed above and below sequence blocks.
...	additional arguments to '.print.fasta.ali'.

Value

Called mostly for its effect but also silently returns block divided concatenated sequence strings as a matrix.

Author(s)

Barry Grant

References

Grant, B.J. et al. (2006) *Bioinformatics* **22**, 2695–2696.

See Also

read.fasta, read.fasta.pdb, pdbaln, seqaln

Examples

```
file <- system.file("examples/kif1a.fa",package="bio3d")
aln  <- read.fasta(file)
print(aln)

# print(aln, col.inds=30:100, numbers=FALSE)
```

`print.xyz` *Printing XYZ coordinates*

Description

Print method for objects of class 'xyz'.

Usage

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

Arguments

`x`	a 'xyz' object indicating 3-D coordinates of biological molecules.
`...`	additional arguments passed to 'print'.

Value

Called for its effect.

Author(s)

Barry Grant

References

Grant, B.J. et al. (2006) *Bioinformatics* **22**, 2695–2696.

See Also

`is.xyz`, `read.ncdf`, `read.pdb`, `read.dcd`, `fit.xyz`

Examples

```
# Read a PDB file
pdb <- read.pdb( system.file("examples/1hel.pdb", package="bio3d") )
print(pdb$xyz)
```

`project.pca` *Project Data onto Principal Components*

Description

Projects data onto principal components.

Usage

```
project.pca(data, pca, angular = FALSE, fit = FALSE, ...)
z2xyz.pca(z.coord, pca)
xyz2z.pca(xyz.coord, pca)
```

Arguments

`data`	a numeric vector or row-wise matrix of data to be projected.
`pca`	an object of class `"pca"` as obtained from functions `pca.xyz` or `pca.tor`.
`angular`	logical, if TRUE the data to be projected is treated as torsion angle data.
`fit`	logical, if TRUE the data is first fitted to `pca$mean`.
`...`	other parameters for `fit.xyz`.
`xyz.coord`	a numeric vector or row-wise matrix of data to be projected.
`z.coord`	a numeric vector or row-wise matrix of PC scores (i.e. the z-scores which are centered and rotated versions of the origional data projected onto the PCs) for conversion to xyz coordinates.

Value

A numeric vector or matrix of projected PC scores.

Author(s)

Karim ElSawy and Barry Grant

References

Grant, B.J. et al. (2006) *Bioinformatics* **22**, 2695–2696.

See Also

`pca.xyz`, `pca.tor`, `fit.xyz`

Examples

```
## Not run:
data(transducin)
attach(transducin, warn.conflicts=FALSE)

gaps.pos <- gap.inspect(pdbs$xyz)

#-- Do PCA without structures 2 and 7
pc.xray <- pca.xyz(pdbs$xyz[-c(2,7), gaps.pos$f.inds])

#-- Project structures 2 and 7 onto the PC space
d <- project.pca(pdbs$xyz[c(2,7), gaps.pos$f.inds], pc.xray)

plot(pc.xray$z[,1], pc.xray$z[,2],col="gray")
points(d[,1],d[,2], col="red")

detach(transducin)

## End(Not run)
```

---

prune.cna *Prune A cna Network Object*

Description

Remove nodes and their associated edges from a cna network graph.

Usage

```
prune.cna(x, edges.min = 1, size.min = 1)
```

Arguments

x	A protein network graph object as obtained from the 'cna' function.
edges.min	A single element numeric vector specifying the minimum number of edges that retained nodes should have. Nodes with less than 'edges.min' will be pruned.
size.min	A single element numeric vector specifying the minimum node size that retained nodes should have. Nodes with less composite residues than 'size.min' will be pruned.

Details

This function is useful for cleaning up cna network plots by removing, for example, small isolated nodes. The output is a new cna object minus the pruned nodes and their associated edges. Node naming is preserved.

Value

A cna class object, see function cna for details.

Note

Some improvements to this function are required, including a better effort to preserve the original community structure rather than calculating a new one. Also may consider removing nodes form the raw.network object that is returned also.

Author(s)

Barry Grant

References

Grant, B.J. et al. (2006) *Bioinformatics* **22**, 2695–2696.

See Also

cna, summary.cna, view.cna, plot.cna

Examples

```
# Load the correlation network
attach(hivp)

# Read the starting PDB file to determine atom correspondence
pdbfile <- system.file("examples/hivp.pdb", package="bio3d")
pdb <- read.pdb(pdbfile)

# Plot coarse grain network based on dynamically coupled communities
par(mfcol=c(1,2), mar=c(0,0,0,0))
plot.cna(net)

# Prune network
dnet <- prune.cna(net, edges.min = 1)
plot(dnet)
```

---

read.all *Read Aligned Structure Data*

Description

Read aligned PDB structures and store their equalvalent atom data, including xyz coordinates, residue numbers, residue type and B-factors.

Usage

```
read.all(aln, prefix = "", pdbext = "", sel = NULL, ...)
```

Arguments

aln	an alignment data structure obtained with read.fasta.
prefix	prefix to aln$id to locate PDB files.
pdbext	the file name extention of the PDB files.
sel	a selection string detailing the atom type data to store (see function store.atom)
...	other parameters for read.pdb.

Details

The input aln, produced with read.fasta, must have identifers (i.e. sequence names) that match the PDB file names. For example the sequence corresponding to the structure file "mypdbdir/1bg2.pdb" should have the identifer 'mypdbdir/1bg2.pdb' or '1bg2' if input 'prefix' and 'pdbext' equal 'mypdbdir/' and 'pdb'. See the examples below.

Sequence miss-matches will generate errors. Thus, care should be taken to ensure that the sequences in the alignment match the sequences in their associated PDB files.

Value

Returns a list of class "pdbs" with the following five components:

xyz	numeric matrix of aligned C-alpha coordinates.
resno	character matrix of aligned residue numbers.
b	numeric matrix of aligned B-factor values.
chain	character matrix of aligned chain identifiers.
id	character vector of PDB sequence/structure names.
ali	character matrix of aligned sequences.
resid	character matrix of aligned 3-letter residue names.
all	numeric matrix of aligned equalvelent atom coordinates.
all.elety	numeric matrix of aligned atom element types.
all.resid	numeric matrix of aligned three-letter residue codes.
all.resno	numeric matrix of aligned residue numbers.

Note

This function is still in development and is NOT part of the offical bio3d package.

The sequence character 'X' is useful for masking unusual or unknown residues, as it can match any other residue type.

Author(s)

Barry Grant

References

Grant, B.J. et al. (2006) *Bioinformatics* **22**, 2695–2696.

See Also

read.fasta, read.pdb, core.find, fit.xyz

Examples

```
# still working on speeding this guy up
cat("\n")
## Not run:
## Read sequence alignment
file <- system.file("examples/kif1a.fa",package="bio3d")
aln  <- read.fasta(file)

## Read aligned PDBs storing all data for 'sel'
sel <- c("N", "CA", "C", "O", "CB", "*G", "*D",  "*E", "*Z")
pdbs <- read.all(aln, sel=sel)

atm <- colnames(pdbs$all)
ca.ind  <- which(atm == "CA")
core <- core.find(pdbs)
core.ind <- c( matrix(ca.ind, nrow=3)[,core$c0.5A.atom] )

## Fit structures
nxyz <- fit.xyz(pdbs$all[1,], pdbs$all,
               fixed.inds  = core.ind,
               mobile.inds = core.ind)

ngap.col <- gap.inspect(nxyz)

#npc.xray <- pca.xyz(nxyz[ ,ngap.col$f.inds])

#a <- mktrj.pca(npc.xray, pc=1, file="pc1-all.pdb",
#               elety=pdbs$all.elety[1,unique( ceiling(ngap.col$f.inds/3) )],
#               resid=pdbs$all.resid[1,unique( ceiling(ngap.col$f.inds/3) )],
#               resno=pdbs$all.resno[1,unique( ceiling(ngap.col$f.inds/3) )] )


## End(Not run)
```

---

read.crd *Read CRD File*

Description

Read a CHARMM CARD (CRD) coordinate file.

Usage

```
read.crd(file, verbose = TRUE)
```

Arguments

file	the name of the CRD file to be read.
verbose	print details of the reading process.

Details

See the function read.pdb for more details.

Value

Returns a list with the following components:

atom	a character matrix containing all atomic coordinate data, with a row per atom and a column per record type. See below for details of the record type naming convention (useful for accessing columns).
xyz	a numeric vector of coordinate data.
calpha	logical vector with length equal to nrow(atom) with TRUE values indicating a C-alpha "elety".

Note

Similar to the output of read.pdb, the column names of atom can be used as a convenient means of data access, namely: Atom serial number "eleno", Atom type "elety", Alternate location indicator "alt", Residue name "resid", Residue sequence number "resno", Code for insertion of residues "insert", Orthogonal coordinates "x", Orthogonal coordinates "y", Orthogonal coordinates "z", Weighting factor "b". See examples for further details.

Author(s)

Barry Grant

References

Grant, B.J. et al. (2006) *Bioinformatics* **22**, 2695–2696.

For a description of CHARMM CARD (CRD) format see:
http://www.charmmtutorial.org/index.php/CHARMM:The_Basics.

See Also

write.crd, read.pdb, atom.select, write.pdb, read.dcd, read.fasta.pdb, read.fasta

Examples

```
## Not run:
 pdb <- read.pdb("1bg2")
 crdfile <- tempfile()
 write.crd(pdb, file=crdfile)
 crd <- read.crd(crdfile)
 ca.inds <- which(crd$calpha)
```

```
 crd$atom[ca.inds[1:20],c("x","y","z")]
# write.pdb(crd, file=tempfile())

## End(Not run)
```

`read.dcd` *Read CHARMM/X-PLOR/NAMD Binary DCD files*

Description

Read coordinate data from a binary DCD trajectory file.

Usage

```
read.dcd(trjfile, big=FALSE, verbose = TRUE, cell = FALSE)
```

Arguments

`trjfile`	name of trajectory file to read. A vector if treat a batch of files
`big`	logical, if TRUE attempt to read large files into a big.matrix object
`verbose`	logical, if TRUE print details of the reading process.
`cell`	logical, if TRUE return cell information only. Otherwise, return coordinates.

Details

Reads a CHARMM or X-PLOR/NAMD binary trajectory file with either big- or little-endian storage formats.

Reading is accomplished with two different sub-functions: `dcd.header`, which reads header info, and `dcd.frame`, which takes header information and reads atoms frame by frame producing an nframes/natom*3 matrix of cartesian coordinates or an nframes/6 matrix of cell parameters.

Value

A numeric matrix of xyz coordinates with a frame/structure per row and a Cartesian coordinate per column or a numeric matrix of cell information with a frame/structure per row and lengths and angles per column.

Note

See CHARMM documentation for DCD format description.

If you experience problems reading your trajectory file with read.dcd() consider first reading your file into VMD and from there exporting a new DCD trajectory file with the 'save coordinates' option. This new file should be easily read with read.dcd().

Error messages beginning 'cannot allocate vector of size' indicate a failure to obtain memory, either because the size exceeded the address-space limit for a process or, more likely, because the system was unable to provide the memory. Note that on a 32-bit OS there may well be enough free memory

available, but not a large enough contiguous block of address space into which to map it. In such cases try setting the input option 'big' to TRUE. This is an experimental option that results in a 'big.matrix' object.

Author(s)

Barry Grant

References

Grant, B.J. et al. (2006) *Bioinformatics* **22**, 2695–2696.

See Also

read.pdb, write.pdb, atom.select

Examples

```
##-- Read cell parameters from example trajectory file
trtfile <- system.file("examples/hivp.dcd", package="bio3d")
trj <- read.dcd(trtfile, cell = TRUE)
##-- Read coordinates from example trajectory file
trj <- read.dcd(trtfile)

## Read the starting PDB file to determine atom correspondence
pdbfile <- system.file("examples/hivp.pdb", package="bio3d")
pdb <- read.pdb(pdbfile)

## select residues 24 to 27 and 85 to 90 in both chains
inds <- atom.select(pdb,"///24:27,85:90///CA/")

## lsq fit of trj on pdb
xyz <- fit.xyz(pdb$xyz, trj, fixed.inds=inds$xyz, mobile.inds=inds$xyz)

##-- RMSD of trj frames from PDB
r1 <- rmsd(a=pdb, b=xyz)


## Not run:
# Pairwise RMSD of trj frames for positions 47 to 54
flap.inds <- atom.select(pdb,"///47:54///CA/")
p <- rmsd(xyz[,flap.inds$xyz])
# plot highlighting flap opening?
plot.dmat(p, color.palette = mono.colors)

## End(Not run)
```

`read.fasta` *Read FASTA formated Sequences*

Description

Read aligned or un-aligned sequences from a FASTA format file.

Usage

```
read.fasta(file, rm.dup = TRUE, to.upper = FALSE, to.dash=TRUE)
```

Arguments

`file`	input sequence file.
`rm.dup`	logical, if TRUE duplicate sequences (with the same names/ids) will be removed.
`to.upper`	logical, if TRUE residues are forced to uppercase.
`to.dash`	logical, if TRUE '.' gap characters are converted to '-' gap characters.

Value

A list with two components:

`ali`	an alignment character matrix with a row per sequence and a column per equivalent aminoacid/nucleotide.
`ids`	sequence names as identifers.
`call`	the matched call.

Note

For a description of FASTA format see: http://www.ncbi.nlm.nih.gov/BLAST/blastcgihelp.shtml. When reading alignment files, the dash '-' is interpreted as the gap character.

Author(s)

Barry Grant

References

Grant, B.J. et al. (2006) *Bioinformatics* **22**, 2695–2696.

See Also

`read.fasta.pdb`

Examples

```
# Read alignment
aln <- read.fasta(system.file("examples/hivp_xray.fa",package="bio3d"))

# Print alignment overview
aln

# Sequence names/ids
head( aln$id )

# Alignment positions 335 to 339
head( aln$ali[,33:39] )

# Sequence d2a4f_b
aa123( aln$ali["d2a4f_b",] )

# Write out positions 33 to 45 only
#aln$ali=aln$ali[,30:45]
#write.fasta(aln, file="eg2.fa")
```

`read.fasta.pdb` *Read Aligned Structure Data*

Description

Read aligned PDB structures and store their C-alpha atom data, including xyz coordinates, residue numbers, residue type and B-factors.

Usage

```
read.fasta.pdb(aln, prefix = "", pdbext = "", fix.ali = FALSE,
             ncore = 1, nseg.scale = 1, ...)
```

Arguments

`aln`	an alignment data structure obtained with `read.fasta`.
`prefix`	prefix to aln$id to locate PDB files.
`pdbext`	the file name extention of the PDB files.
`fix.ali`	logical, if TRUE check consistence between `$ali` and `$resno`, and correct `$ali` if they don't match.
`ncore`	number of CPU cores used to do the calculation. `ncore>1` requires package 'parallel' installed.
`nseg.scale`	split input data into specified number of segments prior to running multiple core calculation. See `fit.xyz`.
`...`	other parameters for `read.pdb`.

Details

The input `aln`, produced with `read.fasta`, must have identifers (i.e. sequence names) that match the PDB file names. For example the sequence corresponding to the structure "1bg2.pdb" should have the identifer '1bg2'. See examples below.

Sequence miss-matches will generate errors. Thus, care should be taken to ensure that the sequences in the alignment match the sequences in their associated PDB files.

Value

Returns a list of class `"pdbs"` with the following five components:

`xyz`	numeric matrix of aligned C-alpha coordinates.
`resno`	character matrix of aligned residue numbers.
`b`	numeric matrix of aligned B-factor values.
`chain`	character matrix of aligned chain identifiers.
`id`	character vector of PDB sequence/structure names.
`ali`	character matrix of aligned sequences.
`resid`	character matrix of aligned 3-letter residue names.
`call`	the matched call.

Note

The sequence character 'X' is useful for masking unusual or unknown residues, as it can match any other residue type.

Author(s)

Barry Grant

References

Grant, B.J. et al. (2006) *Bioinformatics* **22**, 2695–2696.

See Also

`read.fasta`, `read.pdb`, `core.find`, `fit.xyz`, `read.all`

Examples

```
# Read sequence alignment
file <- system.file("examples/kif1a.fa",package="bio3d")
aln  <- read.fasta(file)

# Read aligned PDBs
pdbs <- read.fasta.pdb(aln)
```

```
# Structure/sequence names/ids
basename( pdbs$id )

# Alignment positions 335 to 339
pdbs$ali[,335:339]
pdbs$resid[,335:339]
pdbs$resno[,335:339]
pdbs$b[,335:339]

# Alignment C-alpha coordinates for these positions
pdbs$xyz[, atom2xyz(335:339)]

# See 'fit.xyz()' function for actual coordinate superposition
#  e.g. fit to first structure
# xyz <- fit.xyz(pdbs$xyz[1,], pdbs)
# xyz[, atom2xyz(335:339)]
```

`read.mol2` *Read MOL2 File*

Description

Read a Sybyl MOL2 file

Usage

```
read.mol2(file, maxlines = -1L)
```

Arguments

`file`	a single element character vector containing the name of the MOL2 file to be read.
`maxlines`	the maximum number of lines to read before giving up with large files. Default is all lines.

Details

Basic functionality to parse a MOL2 file. The current version omits bond information, and only `@<TRIPOS>MOLECULE` and `@<TRIPOS>ATOM` records are stored.

In the case of a multi-molecule MOL2 file, each molecule will be stored as an individual object in a list. Conversely, if the multi-molecule MOL2 file contains identical molecules in different conformations (typically a dockin run), then the output will be one object with an `atom` and `xyz` component (xyz in matrix representation; row-wise coordinates).

See examples for further details.

Value

Returns a list of molecules containing the following components:

`atom`	a character matrix containing all atomic coordinate ATOM data, with a row per ATOM and a column per record type. See below for details of the record type naming convention (useful for accessing columns).
`xyz`	a numeric vector or matrix of ATOM coordinate data.
`info`	a numeric vector of MOL2 info data.
`name`	a single element character vector containing the molecule name.

Note

For `atom` list components the column names can be used as a convenient means of data access, namely: Atom serial number "eleno", Atom name "elena", Orthogonal coordinates "x", Orthogonal coordinates "y", Orthogonal coordinates "z", Atom type "elety", Residue name "resid", Atom charge "charge", Status bit "statbit", See examples for further details.

Author(s)

Lars Skjaerven

References

Grant, B.J. et al. (2006) *Bioinformatics* **22**, 2695–2696.

For a description of the MOL2 format see:
http://www.tripos.com/data/support/mol2.pdf.

See Also

`atom.select`, `read.pdb`

Examples

```
cat("\n")
## Not run:
## Read a single entry MOL2 file
## (returns a single object)
mol <- read.mol2("single.mol2")

## ATOM records
mol$atom

## Print some coordinate data
head(mol$atom[, c("x","y","z")])

## Or coordinates as a numeric vector
head(mol$xyz)

## Print atom charges
```

```
head(mol$atom[, "charge")])



## Read a multi-molecule MOL2 file
## (returns a list of objects)
multi.mol <- read.mol2("zinc.mol2")

## Number of molecules described in file
length(multi.mol)

## Access ATOM records for the first molecule
multi.mol[[1]]$atom

## Or coordinates for the second molecule
multi.mol[[2]]$xyz



## Process output from docking (e.g. DOCK)
## (typically one molecule with many conformations)
## (returns one object, but xyz in matrix format)
dock <- read.mol2("dock.mol2")

## Reference PDB file (e.g. X-ray structure)
pdb <- read.pdb("dock_ref.pdb")

## Calculate RMSD of docking modes
sele <- atom.select(dock, "noh")
rmsd(pdb$xyz, dock$xyz, b.inds=sele$xyz)

## End(Not run)
```

---

read.ncdf *Read AMBER Binary netCDF files*

Description

Read coordinate data from a binary netCDF trajectory file.

Usage

```
read.ncdf(trjfile, headonly = FALSE, verbose = TRUE, time = FALSE,
          first = NULL, last = NULL, stride = 1, cell = FALSE,
          at.sel = NULL)
```

Arguments

trjfile	name of trajectory file to read. A vector if treat a batch of files

`headonly`	logical, if TRUE only trajectory header information is returned. If FALSE only trajectory coordinate data is returned.
`verbose`	logical, if TRUE print details of the reading process.
`time`	logical, if TRUE the `first` and `last` have the time unit ps; Otherwise the unit is the frame number.
`first`	starting time or frame number to read; If NULL, start from the begining of the file(s).
`last`	read data until `last` time or frame number; If NULL or equal to -1, read until the end of the file(s).
`stride`	take at every `stride` frame(s)
`cell`	logical, if TRUE and `headonly` is FALSE return cell information only. Otherwise, return header or coordinates.
`at.sel`	an object of class 'select' indicating a subset of atomic coordinates to be read.

Details

Reads a AMBER netCDF format trajectory file with the help of David W. Pierce's (UCSD) ncdf package available from CRAN.

Value

A list of trajectory header data, a numeric matrix of xyz coordinates with a frame/structure per row and a Cartesian coordinate per column, or a numeric matrix of cell information with a frame/structure per row and lengths and angles per column. If time=TRUE, row names of returned coordinates or cell are set to be the physical time of corresponding frames.

Note

See AMBER documentation for netCDF format description.

NetCDF binary trajectory files are supported by the AMBER modules sander, pmemd and ptraj. Compared to formatted trajectory files, the binary trajectory files are smaller, higher precision and significantly faster to read and write.

NetCDF provides for file portability across architectures, allows for backwards compatible extensibility of the format and enables the files to be self-describing. Support for this format is available in VMD.

If you experience problems reading your trajectory file with read.ncdf() consider first reading your file into VMD and from there exporting a new DCD trajectory file with the 'save coordinates' option. This new file should be easily read with read.dcd().

Author(s)

Barry Grant

References

Grant, B.J. et al. (2006) *Bioinformatics* **22**, 2695–2696. http://www.unidata.ucar.edu/packages/netcdf/ http://cirrus.ucsd.edu/~pierce/ncdf/ http://ambermd.org/netcdf/nctraj.html

See Also

read.dcd, write.ncdf, read.pdb, write.pdb, atom.select

Examples

```
## Not run:
##-- Read example trajectory file
trtfile <- system.file("examples/hivp.dcd", package="bio3d")
trj <- read.dcd(trtfile)

## Write to netCDF format
write.ncdf(trj, "newtrj.nc")

## Read trj
trj <- read.ncdf("newtrj.nc")

## End(Not run)
```

---

read.pdb *Read PDB File*

Description

Read a Protein Data Bank (PDB) coordinate file.

Usage

```
read.pdb(file, maxlines = -1, multi = FALSE, rm.insert = FALSE,
         rm.alt = TRUE, verbose = TRUE)
## S3 method for class 'pdb'
print(x, ...)
## S3 method for class 'pdb'
summary(object, printseq=FALSE, ...)
```

Arguments

file	a single element character vector containing the name of the PDB file to be read, or the four letter PDB identifier for online file access.
maxlines	the maximum number of lines to read before giving up with large files. By default if will read up to the end of input on the connection.
multi	logical, if TRUE multiple ATOM records are read for all models in multi-model files and their coordinates returned.
rm.insert	logical, if TRUE PDB insert records are ignored.
rm.alt	logical, if TRUE PDB alternate records are ignored.
verbose	print details of the reading process.
x	a PDB structure object obtained from read.pdb.

object	a PDB structure object obtained from read.pdb.
printseq	logical, if TRUE the PDB ATOM sequence will be printed to the screen. See also pdbseq.
...	additional arguments to 'print'.

Details

maxlines may be set so as to restrict the reading to a portion of input files. Note that the preferred means of reading large multi-model files is via binary DCD or NetCDF format trajectory files (see the read.dcd and read.ncdf functions).

Value

Returns a list of class "pdb" with the following components:

atom	a data.frame containing all atomic coordinate ATOM and HETATM data, with a row per ATOM/HETATM and a column per record type. See below for details of the record type naming convention (useful for accessing columns).
helix	'start', 'end' and 'length' of H type sse, where start and end are residue numbers "resno".
sheet	'start', 'end' and 'length' of E type sse, where start and end are residue numbers "resno".
seqres	sequence from SEQRES field.
xyz	a numeric vector (or matrix for multi-model PDB files) of ATOM and HETATM coordinate data.
calpha	logical vector with length equal to nrow(atom) with TRUE values indicating a C-alpha "elety".
call	the matched call.

Note

For both atom and het list components the column names can be used as a convenient means of data access, namely: Atom serial number "eleno" , Atom type "elety", Alternate location indicator "alt", Residue name "resid", Chain identifier "chain", Residue sequence number "resno", Code for insertion of residues "insert", Orthogonal coordinates "x", Orthogonal coordinates "y", Orthogonal coordinates "z", Occupancy "o", and Temperature factor "b". See examples for further details.

Author(s)

Barry Grant

References

Grant, B.J. et al. (2006) *Bioinformatics* **22**, 2695–2696.

For a description of PDB format (version3.3) see:
http://www.wwpdb.org/documentation/format33/v3.3.html.

See Also

atom.select, write.pdb, read.dcd, read.fasta.pdb, read.fasta

Examples

```
## Read a PDB file from the RCSB online database
#pdb <- read.pdb("4q21")

## Read a PDB file from those included with the package
pdb <- read.pdb( system.file("examples/1hel.pdb", package="bio3d") )

## Print a brief composition summary
pdb

## Examine the storage format (or internal *str*ucture)
str(pdb)

## Print data for the first four atom
pdb$atom[1:4,]

## Print some coordinate data
head(pdb$atom[, c("x","y","z")])

## Or coordinates as a numeric vector
head(pdb$xyz)

## Print C-alpha coordinates (can also use 'atom.select' function)
head(pdb$atom[pdb$calpha, c("resid","elety","x","y","z")])
inds <- atom.select(pdb, elety="CA")
head( pdb$atom[inds$atom, ] )

## The atom.select() function returns 'indices' (row numbers)
## that can be used for accessing subsets of PDB objects, e.g.
inds <- atom.select(pdb,"ligand")
pdb$atom[inds$atom,]
pdb$xyz[inds$xyz]

## See the help page for atom.select() function for more details.


## Not run:
## Print SSE data for helix and sheet,
##  see also dssp() and stride() functions
print.sse(pdb)
pdb$helix
pdb$sheet$start

## Print SEQRES data
pdb$seqres

## SEQRES as one letter code
aa321(pdb$seqres)
```

```
## Where is the P-loop motif in the ATOM sequence
inds.seq <- motif.find("G....GKT", pdbseq(pdb))
pdbseq(pdb)[inds.seq]

## Where is it in the structure
inds.pdb <- atom.select(pdb,resno=inds.seq, elety="CA")
pdb$atom[inds.pdb$atom,]
pdb$xyz[inds.pdb$xyz]

## View in interactive 3D mode
#view(pdb)

## End(Not run)
```

`read.pdcBD` *Read PQR output from pdcBD File*

Description

Read a pdcBD PQR coordinate file.

Usage

```
read.pdcBD(file, maxlines = 50000, multi = FALSE, rm.insert = FALSE,
         rm.alt = TRUE, verbose = TRUE)
```

Arguments

`file`	the name of the pdcBD PQR file to be read.
`maxlines`	the maximum number of lines to read before giving up with large files. Default is 50,000 lines.
`multi`	logical, if TRUE multiple ATOM records are read for all models in multi-model files.
`rm.insert`	logical, if TRUE PDB insert records are ignored.
`rm.alt`	logical, if TRUE PDB alternate records are ignored.
`verbose`	print details of the reading process.

Details

`maxlines` may require increasing for some large multi-model files. The preferred means of reading such data is via binary DCD format trajectory files (see the `read.dcd` function).

Value

Returns a list of class "pdb" with the following components:

atom	a character matrix containing all atomic coordinate ATOM data, with a row per ATOM and a column per record type. See below for details of the record type naming convention (useful for accessing columns).
het	a character matrix containing atomic coordinate records for atoms within "non-standard" HET groups (see atom).
helix	'start', 'end' and 'length' of H type sse, where start and end are residue numbers "resno".
sheet	'start', 'end' and 'length' of E type sse, where start and end are residue numbers "resno".
seqres	sequence from SEQRES field.
xyz	a numeric vector of ATOM coordinate data.
calpha	logical vector with length equal to nrow(atom) with TRUE values indicating a C-alpha "elety".

Note

For both atom and het list components the column names can be used as a convenient means of data access, namely: Atom serial number "eleno" , Atom type "elety", Alternate location indicator "alt", Residue name "resid", Chain identifier "chain", Residue sequence number "resno", Code for insertion of residues "insert", Orthogonal coordinates "x", Orthogonal coordinates "y", Orthogonal coordinates "z", Occupancy "o", and Temperature factor "b". See examples for further details.

Author(s)

Barry Grant

References

Grant, B.J. et al. (2006) *Bioinformatics* **22**, 2695–2696.

For a description of PDB format (version3.3) see:
http://www.wwpdb.org/documentation/format33/v3.3.html.

See Also

atom.select, write.pdb, read.dcd, read.fasta.pdb, read.fasta

Examples

```
# Read a PDB file
pdb <- read.pdb( "1bg2" )

# Print data for the first atom
pdb$atom[1,]
# Look at the first het atom
```

```
pdb$het[1,]
# Print some coordinate data
pdb$atom[1:20, c("x","y","z")]

# Print C-alpha coordinates (can also use 'atom.select')
##pdb$xyz[pdb$calpha, c("resid","x","y","z")]

# Print SSE data (for helix and sheet)
pdb$helix
pdb$sheet$start

# Print SEQRES data
pdb$seqres

# Renumber residues
nums <- as.numeric(pdb$atom[,"resno"])
pdb$atom[,"resno"] <- nums - (nums[1] - 1)

# Write out renumbered PDB file
#write.pdb(pdb=pdb,file="eg.pdb")
```

`read.pqr` *Read PQR File*

Description

Read a PQR coordinate file.

Usage

```
read.pqr(file, maxlines = -1, multi = FALSE, rm.insert = FALSE,
         rm.alt = TRUE, verbose = TRUE)
```

Arguments

`file`	the name of the PQR file to be read.
`maxlines`	the maximum number of lines to read before giving up with large files. By default if will read up to the end of input on the connection.
`multi`	logical, if TRUE multiple ATOM records are read for all models in multi-model files.
`rm.insert`	logical, if TRUE PDB insert records are ignored.
`rm.alt`	logical, if TRUE PDB alternate records are ignored.
`verbose`	print details of the reading process.

Details

`maxlines` may require increasing for some large multi-model files. The preferred means of reading such data is via binary DCD format trajectory files (see the `read.dcd` function).

Value

Returns a list of class "pdb" with the following components:

atom	a character matrix containing all atomic coordinate ATOM data, with a row per ATOM and a column per record type. See below for details of the record type naming convention (useful for accessing columns).
het	a character matrix containing atomic coordinate records for atoms within "non-standard" HET groups (see atom).
helix	'start', 'end' and 'length' of H type sse, where start and end are residue numbers "resno".
sheet	'start', 'end' and 'length' of E type sse, where start and end are residue numbers "resno".
seqres	sequence from SEQRES field.
xyz	a numeric vector of ATOM coordinate data.
calpha	logical vector with length equal to nrow(atom) with TRUE values indicating a C-alpha "elety".

Note

For both atom and het list components the column names can be used as a convenient means of data access, namely: Atom serial number "eleno" , Atom type "elety", Alternate location indicator "alt", Residue name "resid", Chain identifier "chain", Residue sequence number "resno", Code for insertion of residues "insert", Orthogonal coordinates "x", Orthogonal coordinates "y", Orthogonal coordinates "z", Occupancy "o", and Temperature factor "b". See examples for further details.

Author(s)

Barry Grant

References

Grant, B.J. et al. (2006) *Bioinformatics* **22**, 2695–2696.

For a description of PDB format (version3.3) see:
http://www.wwpdb.org/documentation/format33/v3.3.html.

See Also

atom.select, write.pdb, read.dcd, read.fasta.pdb, read.fasta

Examples

```
# Read a PDB file
pdb <- read.pdb( "4q21" )

# Print data for the first atom
pdb$atom[1,]
# Look at the first het atom
```

```
pdb$het[1,]
# Print some coordinate data
pdb$atom[1:20, c("x","y","z")]

# Print C-alpha coordinates (can also use 'atom.select')
##pdb$xyz[pdb$calpha, c("resid","x","y","z")]

# Print SSE data (for helix and sheet)
pdb$helix
pdb$sheet$start

# Print SEQRES data
pdb$seqres

# Renumber residues
nums <- as.numeric(pdb$atom[,"resno"])
pdb$atom[,"resno"] <- nums - (nums[1] - 1)

# Write out renumbered PDB file
#write.pdb(pdb=pdb,file="eg.pdb")
```

`rgyr` *Radius of Gyration*

Description

Calculate the radius of gyration of coordinate sets.

Usage

```
rgyr(xyz, mass=NULL, ncore=1, nseg.scale=1)
```

Arguments

`xyz`	a numeric vector, matrix or list object with an `xyz` component, containing one or more coordinate sets.
`mass`	a numeric vector of atomic masses (unit a.m.u.), or a PDB object with masses stored in the "B-factor" column. If `mass==NULL`, all atoms are assumed carbon.
`ncore`	number of CPU cores used to do the calculation. `ncore>1` requires package 'parallel' installed.
`nseg.scale`	split input data into specified number of segments prior to running multiple core calculation. See `fit.xyz`.

Details

Radius of gyration is a standard measure of overall structural change of macromolecules.

Value

Returns a numeric vector of radius of gyration.

Author(s)

Xin-Qiu Yao & Pete Kekenes-Huskey

References

Grant, B.J. et al. (2006) *Bioinformatics* **22**, 2695–2696.

See Also

`fit.xyz`, `rmsd`, `read.pdb`, `read.fasta.pdb`

Examples

```
# -- Calculate Rog of single structure
pdb <- read.pdb("1bg2")
mass <- rep(12, length(pdb$xyz)/3)
mass[substr(pdb$atom[,"elety"], 1, 1) == "N"] <- 14
mass[substr(pdb$atom[,"elety"], 1, 1) == "H"] <- 1
mass[substr(pdb$atom[,"elety"], 1, 1) == "O"] <- 16
mass[substr(pdb$atom[,"elety"], 1, 1) == "S"] <- 32

rgyr(pdb, mass)


## Not run:
# -- Calculate Rog of a trajectory
xyz <- read.dcl(system.file("examples/hivp.dcd", package="bio3d"))
rg <- rgyr(xyz)
rg[1:10]


## End(Not run)
```

---

rle2 *Run Length Encoding with Indices*

Description

Compute the lengths, values and indices of runs of equal values in a vector. This is a modifed version of base function `rle()`.

Usage

```
rle2(x)

## S3 method for class 'rle2'
print(x, digits = getOption("digits"), prefix = "", ...)
```

Arguments

x	an atomic vector for rle(); an object of class "rle" for inverse.rle().
...	further arguments; ignored here.
digits	number of significant digits for printing, see print.default.
prefix	character string, prepended to each printed line.

Details

Missing values are regarded as unequal to the previous value, even if that is also missing.

inverse.rle() is the inverse function of rle2() and rle(), reconstructing x from the runs.

Value

rle() returns an object of class "rle" which is a list with components:

lengths	an integer vector containing the length of each run.
values	a vector of the same length as lengths with the corresponding values.

Examples

```
x <- rev(rep(6:10, 1:5))
rle(x)
## lengths [1:5]  5 4 3 2 1
## values  [1:5] 10 9 8 7 6
rle2
## lengths: int [1:5] 5 4 3 2 1
## values : int [1:5] 10 9 8 7 6
## indices: int [1:5] 5 9 12 14 15
```

---

rmsd *Root Mean Square Deviation*

Description

Calculate the RMSD between coordinate sets.

Usage

```
rmsd(a, b=NULL, a.inds=NULL, b.inds=NULL, fit=FALSE, ncore=1, nseg.scale=1)
```

Arguments

a	a numeric vector containing the reference coordinate set for comparison with the coordinates in b. Alternatively, if b=NULL then a can be a matrix or list object containing multiple coordinates for pairwise comparison.
b	a numeric vector, matrix or list object with an xyz component, containing one or more coordinate sets to be compared with a.
a.inds	a vector of indices that selects the elements of a upon which the calculation should be based.
b.inds	a vector of indices that selects the elements of b upon which the calculation should be based.
fit	logical, if TRUE coordinate superposition is performed prior to RMSD calculation.
ncore	number of CPU cores used to do the calculation. ncore>1 requires package 'parallel' installed.
nseg.scale	split input data into specified number of segments prior to running multiple core calculation. See fit.xyz.

Details

RMSD is a standard measure of structural distance between coordinate sets.

Structure a[a.inds] and b[b.inds] should have the same length.

A least-squares fit is performed prior to RMSD calculation by setting fit=TRUE. See the function fit.xyz for more details of the fitting process.

Value

Returns a numeric vector of RMSD value(s).

Author(s)

Barry Grant

References

Grant, B.J. et al. (2006) *Bioinformatics* **22**, 2695–2696.

See Also

fit.xyz, rot.lsq, read.pdb, read.fasta.pdb

Examples

```
# -- Calculate RMSD between two or more structures
aln <- read.fasta(system.file("examples/kif1a.fa",package="bio3d"))
pdbs <- read.fasta.pdb(aln)
```

```
# Indices
gaps <- unique( which(is.na(pdbs$xyz),arr.ind=TRUE)[,2] )
inds <- c(1:ncol(pdbs$xyz))[-gaps]

# RMSD between structure 1 and structure 2
rmsd(a=pdbs$xyz[1,], b=pdbs$xyz[2,], a.inds=inds, b.inds=inds)
rmsd(a=pdbs$xyz[1,], b=pdbs$xyz[2,], a.inds=inds, b.inds=inds, fit=TRUE)


## Not run:
# RMSD between structure 1 and structures 2 and 3
rmsd(a=pdbs$xyz[1,], b=pdbs$xyz[2:3,], a.inds=inds, b.inds=inds)

# RMSD between structure 1 and all structures in alignment
rmsd(a=pdbs$xyz[1,], b=pdbs, a.inds=inds, b.inds=inds, fit=TRUE)

# pairwise RMSD
rmsd(pdbs$xyz[,inds])

## End(Not run)
```

`rmsd.filter` *RMSD Filter*

Description

Identify and filter subsets of conformations at a given RMSD cutoff.

Usage

```
rmsd.filter(xyz = NULL, rmsd.mat = NULL, cutoff = 0.5,
            fit = TRUE, verbose = TRUE, inds = NULL,
            ncore = 1, nseg.scale = 1)
```

Arguments

`xyz`	a numeric matrix or list object containing multiple coordinates for pairwise comparison, such as that obtained from `read.fasta.pdb`. Not used if `rmsd.mat` is given.
`rmsd.mat`	an optional matrix of RMSD values obtained from `rmsd`.
`cutoff`	a numeric rmsd cutoff value.
`fit`	logical, if TRUE coordinate superposition is performed prior to RMSD calculation.
`verbose`	logical, if TRUE progress details are printed.
`inds`	a vector of indices that selects the elements of `xyz` upon which the calculation should be based. By default, all the non-gap sites in `xyz`.

ncore	number of CPU cores used to do the calculation. ncore>1 requires package 'parallel' installed.
nseg.scale	split input data into specified number of segments prior to running multiple core calculation. See fit.xyz.

Details

This function performs hierarchical cluster analysis of a given matrix of RMSD values 'rmsd.mat', or an RMSD matrix calculated from a given coordinate matrix 'xyz', to identify conformers that fall below a given RMSD cutoff value 'cutoff'.

Value

Returns a list object with components:

ind	indices of the conformers (rows) below the cutoff value.
tree	an object of class "hclust", which describes the tree produced by the clustering process.
rmsd.mat	a numeric matrix with all pairwise RMSD values.

Author(s)

Barry Grant

References

Grant, B.J. et al. (2006) *Bioinformatics* **22**, 2695–2696.

See Also

rmsd, read.pdb, read.fasta.pdb, read.dcd

Examples

```
## Not run:
data(kinesin)
attach(kinesin, warn.conflicts=FALSE)
k <- rmsd.filter(xyz=pdbs,cutoff=0.5)
pdbs$id[k$ind]
plot(k$tree, ylab="RMSD")
abline(h=0.5, col="gray")

detach(kinesin)

## End(Not run)
```

rmsf *Atomic RMS Fluctuations*

Description

Calculate atomic root mean squared fluctuations.

Usage

```
rmsf(xyz)
```

Arguments

xyz	numeric matrix of coordinates with each row corresponding to an individual conformer.

Details

An often used measure of conformational variance.

Value

Returns a numeric vector of RMSF values.

Author(s)

Barry Grant

References

Grant, B.J. et al. (2006) *Bioinformatics* **22**, 2695–2696.

See Also

read.dcd, fit.xyz, read.fasta.pdb

Examples

```
data(transducin)
attach(transducin, warn.conflicts=FALSE)

# Ignore Gaps
gaps <- gap.inspect(pdbs$ali)

r <- rmsf(pdbs$xyz)
plot( r[gaps$f.inds], typ="h", ylab="RMSF (A)")

detach(transducin)
```

rmsip *Root Mean Square Inner Product*

Description

Calculate the RMSIP between two mode subspaces.

Usage

```
rmsip(...)
## S3 method for class 'enma'
rmsip(enma, ncore=NULL, subset=10, ...)
## Default S3 method:
rmsip(modes.a, modes.b, subset=10,
                        row.name="a", col.name="b", ...)
```

Arguments

enma	an object of class "enma" obtained from function nma.pdbs.
ncore	number of CPU cores used to do the calculation. ncore>1 requires package 'parallel' installed.
subset	the number of modes to consider.
modes.a	an object of class "pca" or "nma" as obtained from functions pca.xyz or nma.
modes.b	an object of class "pca" or "nma" as obtained from functions pca.xyz or nma.
row.name	prefix name for the rows.
col.name	prefix name for the columns.
...	arguments passed to associated functions.

Details

RMSIP is a measure for the similarity between two set of modes obtained from principal component or normal modes analysis.

Value

Returns an rmsip object with the following components:

overlap	a numeric matrix containing pairwise (squared) dot products between the modes.
rmsip	a numeric RMSIP value.

For function rmsip.enma a numeric matrix containing all pairwise RMSIP values of the modes stored in the enma object.

Author(s)

Lars Skjaerven

References

Grant, B.J. et al. (2006) *Bioinformatics* **22**, 2695–2696. Amadei, A. et al. (1999) *Proteins* **36**, 19–424.

See Also

pca, nma, overlap.

Other similarity measures: sip, covsoverlap, bhattacharyya.

Examples

```
## Not run:
# Load data for HIV example
trj <- read.dcd(system.file("examples/hivp.dcd", package="bio3d"))
pdb <- read.pdb(system.file("examples/hivp.pdb", package="bio3d"))

# Do PCA on simulation data
xyz.md <- fit.xyz(pdb$xyz, trj, fixed.inds=1:ncol(trj))
pc.sim <- pca.xyz(xyz.md)

# NMA
modes <- nma(pdb)

# Calculate the RMSIP between the MD-PCs and the NMA-MODEs
r <- rmsip(modes, pc.sim, subset=10, row.name="NMA", col.name="PCA")

# Plot pairwise overlap values
plot(r, xlab="NMA", ylab="PCA")

## End(Not run)
```

---

sdENM *Index for the sdENM ff*

Description

A dictonary of spring force constants for the sdENM force field.

Usage

```
data(sdENM)
```

Format

An array of 27 matrices containg the spring force constants for the 'sdENM' force field (see Dehouch et al for more information). Each matrix in the array holds the force constants for all amino acid pairs for a specific distance range.

See examples for more details.

Source

Dehouck Y. & Mikhailov A.S. (2013) *PLoS Comput Biol* **9**:e1003209.

References

Grant, B.J. et al. (2006) *Bioinformatics* **22**, 2695–2696.

Examples

```
## Load force constant data
data(sdENM)

## force constants for amino acids A, C, D, E, and F
## in distance range [4, 4.5)
sdENM[1:5, 1:5, 1]

## and distance range [4.5, 5)
sdENM[1:5, 1:5, 2]

## amino acid pair A-P, at distance 4.2
sdENM["A", "P", 1]

## Not run:
## for use in NMA
pdb <- read.pdb( system.file("examples/1hel.pdb", package="bio3d") )
modes <- nma(pdb, ff="sdenm")

## End(Not run)
```

---

seq2aln *Add a Sequence to an Existing Alignmnet*

Description

Add one or more sequences to an existing multiple alignment that you wish to keep intact.

Usage

```
seq2aln(seq2add, aln, id = "seq", exefile = "muscle", file = "aln.fa")
```

Arguments

seq2add	an sequence character vector or an alignment list object with id and ali components, similar to that generated by read.fasta and seqaln.
aln	an alignment list object with id and ali components, similar to that generated by read.fasta and seqaln.
id	a vector of sequence names to serve as sequence identifers.

exefile	file path to the 'MUSCLE' program on your system (i.e. how is 'MUSCLE' invoked).
file	name of 'FASTA' output file to which alignment should be written.

Details

This function calls the 'MUSCLE' program, to perform a profile profile alignment, which MUST BE INSTALLED on your system and in the search path for executables.

Value

A list with two components:

ali	an alignment character matrix with a row per sequence and a column per equivalent aminoacid/nucleotide.
id	sequence names as identifers.

Note

A system call is made to the 'MUSCLE' program, which must be installed on your system and in the search path for executables.

Author(s)

Barry Grant

References

Grant, B.J. et al. (2006) *Bioinformatics* **22**, 2695–2696.

'MUSCLE' is the work of Edgar: Edgar (2004) *Nuc. Acid. Res.* **32**, 1792–1797.

Full details of the 'MUSCLE' algorithm, along with download and installation instructions can be obtained from:
http://www.drive5.com/muscle.

See Also

seqaln, read.fasta, read.fasta.pdb, seqbind

Examples

```
## Not run:
aa.1 <- pdbseq( read.pdb("1bg2") )
aa.2 <- pdbseq( read.pdb("3dc4") )
aa.3 <- pdbseq( read.pdb("1mkj") )

aln <- seqaln( seqbind(aa.1,aa.2) )

seq2aln(aa.3, aln)
```

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

---

seqaln *Sequence Alignment with MUSCLE*

Description

Create multiple alignments of amino acid or nucleotide sequences according to the method of Edgar.

Usage

```
seqaln(aln, id=NULL, profile=NULL, exefile="muscle", outfile="aln.fa",
       protein=TRUE, seqgroup=FALSE, refine=FALSE, extra.args="",
       verbose=FALSE)
```

Arguments

aln	a sequence character matrix, as obtained from seqbind, or an alignment list object as obtained from read.fasta.
id	a vector of sequence names to serve as sequence identifers.
profile	a profile alignment of class 'fasta' (e.g. obtained from read.fasta). The alignment aln will be added to the profile.
exefile	file path to the 'MUSCLE' program on your system (i.e. how is 'MUSCLE' invoked). Alternatively, 'CLUSTALO' can be used.
outfile	name of 'FASTA' output file to which alignment should be written.
protein	logical, if TRUE the input sequences are assumed to be protein not DNA or RNA.
seqgroup	logical, if TRUE similar sequences are grouped together in the output.
refine	logical, if TRUE the input sequences are assumed to already be aligned, and only tree dependent refinement is performed.
extra.args	a single character string containing extra command line arguments for the alignment program.
verbose	logical, if TRUE 'MUSCLE' warning and error messages are printed.

Details

Sequence alignment attempts to arrange the sequences of protein, DNA or RNA, to highlight regions of shared similarity that may reflect functional, structural, and/or evolutionary relationships between the sequences.

Aligned sequences are represented as rows within a matrix. Gaps ('-') are inserted between the aminoacids or nucleotides so that equivalent characters are positioned in the same column.

This function calls the 'MUSCLE' program, to perform a multiple sequence alignment, which MUST BE INSTALLED on your system and in the search path for executables.

If you have a large number of input sequences (a few thousand), or they are very long, the default settings may be too slow for practical use. A good compromise between speed and accuracy is to run just the first two iterations of the 'MUSCLE' algorithm by setting the `extra.args` argument to "-maxiters 2".

You can set 'MUSCLE' to improve an existing alignment by setting `refine` to TRUE.

To inspect the sequence clustering used by 'MUSCLE' to produce alignments, include "-tree2 tree.out" in the `extra.args` argument. You can then load the "tree.out" file with the 'read.tree' function from the 'ape' package.

'CLUSTALO' can be used as an alternative to 'MUSCLE' by specifiying `exefile='clustalo'`. This might be useful e.g. when adding several sequences to a profile alignment.

Value

A list with two components:

`ali`	an alignment character matrix with a row per sequence and a column per equivalent aminoacid/nucleotide.
`ids`	sequence names as identifers.
`call`	the matched call.

Note

A system call is made to the 'MUSCLE' program, which must be installed on your system and in the search path for executables.

Author(s)

Barry Grant

References

Grant, B.J. et al. (2006) *Bioinformatics* **22**, 2695–2696.

'MUSCLE' is the work of Edgar: Edgar (2004) *Nuc. Acid. Res.* **32**, 1792–1797.

Full details of the 'MUSCLE' algorithm, along with download and installation instructions can be obtained from:
http://www.drive5.com/muscle.

See Also

`read.fasta`, `read.fasta.pdb`, `get.seq`, `seqbind`, `pdbaln`, `plot.fasta`, `blast.pdb`

Examples

```
## Not run:
##-- Basic sequence alignemnt
seqs <- get.seq(c("4q21_A", "1ftn_A"))
aln <- seqaln(seqs)
```

```
##-- add a sequence to the (profile) alignment
seq <- get.seq("1tnd_A")
aln <- seqaln(seq, profile=aln)


##-- Read a folder/directory of PDB files
#pdb.path <- "my_dir_of_pdbs"
#files  <- list.files(path=pdb.path ,
#                     pattern=".pdb",
#                     full.names=TRUE)

##-- Use online files
files <- get.pdb(c("4q21","1ftn"), URLonly=TRUE)

##-- Extract and store sequences
raw <- NULL
for(i in 1:length(files)) {
  pdb <- read.pdb(files[i])
  raw <- seqbind(raw, pdbseq(pdb) )
}

##-- Align these sequences
aln <- seqaln(raw, id=files, outfile="seqaln.fa")

##-- Read Aligned PDBs storing coordinate data
pdbs <- read.fasta.pdb(aln)

## Sequence identity
seqidentity(aln)

## Note that all the above can be done with the pdbaln() function:
#pdbs <- pdbaln(files)


##-- For identical sequences with masking use a custom matrix
aln <- list(ali=seqbind(c("X","C","X","X","A","G","K"),
                        c("C","-","A","X","G","X","X","K")),
            id=c("a","b"))

temp <- seqaln(aln=aln, outfile="temp.fas", protein=TRUE,
               extra.args= paste("-matrix",
                system.file("matrices/custom.mat", package="bio3d"),
                "-gapopen -3.0 ",
                "-gapextend -0.5",
                "-center 0.0") )


## End(Not run)
```

---

seqaln.pair *Sequence Alignment of Identical Protein Sequences*

Description

Create multiple alignments of amino acid sequences according to the method of Edgar.

Usage

```
seqaln.pair(aln, extra.args = "", ...)
```

Arguments

aln	a sequence character matrix, as obtained from seqbind, or an alignment list object as obtained from read.fasta.
extra.args	a single character string containing extra command line arguments for the alignment program.
...	additional arguments for the function seqaln.

Details

This function is intended for the alignment of identical sequences only. For standard alignment see the related function seqaln.

This function is useful for determining the equivalences between sequences and structures. For example in aligning a PDB sequence to an existing multiple sequence alignment, where one would first mask the alignment sequences and then run the alignment to determine equivalences.

Value

A list with two components:

ali	an alignment character matrix with a row per sequence and a column per equivalent aminoacid/nucleotide.
ids	sequence names as identifers.

Note

A system call is made to the 'MUSCLE' program, which must be installed on your system and in the search path for executables.

Author(s)

Barry Grant

References

Grant, B.J. et al. (2006) *Bioinformatics* **22**, 2695–2696.

'MUSCLE' is the work of Edgar: Edgar (2004) *Nuc. Acid. Res.* **32**, 1792–1797.

Full details of the 'MUSCLE' algorithm, along with download and installation instructions can be obtained from:
http://www.drive5.com/muscle.

See Also

seqaln, read.fasta, read.fasta.pdb, seqbind

Examples

```
## NOTE: FOLLOWING EXAMPLE NEEDS MUSCLE INSTALLED
if(check.utility("muscle")) {

##- Aligning a PDB sequence to an existing sequence alignment


##- Simple example
aln <- list(ali=seqbind(c("X","C","X","X","A","G","K"),
                        c("C","-","A","X","G","X","X","K")),
            id=c("a","b"))

seqaln.pair(aln, outfile = tempfile())

}
```

---

seqbind *Combine Sequences by Rows Without Recycling*

Description

Take vectors and/or matrices arguments and combine them row-wise without recycling them (as is the case with rbind).

Usage

```
seqbind(..., blank = "-")
```

Arguments

...	vectors, matrices, and/or alignment 'fasta' objects to combine.
blank	a character to add to short arguments, to achieve the same length as the longer argument.

Value

A matrix combining input arguments row-wise.

Author(s)

Barry Grant

References

Grant, B.J. et al. (2006) *Bioinformatics* **22**, 2695–2696.

See Also

seqaln, read.fasta, read.pdb, write.fasta, rbind

Examples

```
## Not run:
## Read two pdbs
a.pdb <- read.pdb("1bg2")
b.pdb <- read.pdb("1goj")

seqs <- seqbind(aa321(a.pdb$atom[a.pdb$calpha,"resid"]),
                aa321(b.pdb$atom[b.pdb$calpha,"resid"]))

# seqaln(seqs)

## End(Not run)
```

---

seqidentity *Percent Identity*

Description

Determine the percent identity scores for aligned sequences.

Usage

```
seqidentity(alignment, normalize=TRUE, similarity=FALSE, ncore=1, nseg.scale=1)
```

Arguments

alignment	sequence alignment obtained from read.fasta or an alignment character matrix.
normalize	logical, if TRUE output is normalized to values between 0 and 1 otherwise percent identity is returned.
similarity	logical, if TRUE sequence similarity is calculated instead of identity.
ncore	number of CPU cores used to do the calculation. ncore>1 requires package 'parallel' installed.
nseg.scale	split input data into specified number of segments prior to running multiple core calculation. See fit.xyz.

Details

The percent identity value is a single numeric score determined for each pair of aligned sequences. It measures the number of identical residues ("matches") in relation to the length of the alignment.

Value

Returns a numeric matrix with all pairwise identity values.

Author(s)

Barry Grant

References

Grant, B.J. et al. (2006) *Bioinformatics* **22**, 2695–2696.

See Also

read.fasta, ide.filter, entropy, consensus

Examples

```
## Not run:
aln <- read.fasta( system.file("examples/kif1a.fa",
                     package = "bio3d") )

## End(Not run)

data(kinesin)
attach(kinesin, warn.conflicts=FALSE)

ide.mat <- seqidentity(pdbs)

# Plot identity matrix
plot.dmat(ide.mat, color.palette=mono.colors,
          main="Sequence Identity", xlab="Structure No.",
          ylab="Structure No.")

# Histogram of pairwise identity values
hist(ide.mat[upper.tri(ide.mat)], breaks=30,xlim=c(0,1),
     main="Sequence Identity", xlab="Identity")

# Compare two sequences
seqidentity( rbind(pdbs$ali[1,], pdbs$ali[20,]) )

detach(kinesin)
```

---

setup.ncore *Setup for Running Bio3D Functions using Multiple CPU Cores*

Description

Internally used in parallelized Bio3D functions.

Usage

```
setup.ncore(ncore, bigmem = FALSE)
```

Arguments

ncore	User set (or default) value of 'ncore'.
bigmem	logical, if TRUE also check the availability of 'bigmemory' package.

Details

Check packages and set correct value of 'ncore'.

Value

The actual value of 'ncore'.

Examples

```
   setup.ncore(NULL)
   setup.ncore(1)
#   setup.ncore(2)
```

---

sip *Square Inner Product*

Description

Calculate the correlation between two atomic fluctuation vectors.

Usage

```
sip(...)
## S3 method for class 'nma'
sip(a, b, ...)
## S3 method for class 'enma'
sip(enma, ncore=NULL, ...)
## Default S3 method:
sip(v, w, ...)
```

Arguments

enma	an object of class "enma" obtained from function nma.pdbs.
ncore	number of CPU cores used to do the calculation. ncore>1 requires package 'parallel' installed.
a	an 'nma' object as object from function nma to be compared to b.
b	an 'nma' object as object from function nma to be compared to a.

v	a numeric vector containing the atomic fluctuation values.
w	a numeric vector containing the atomic fluctuation values.
...	arguments passed to associated functions.

Details

SIP is a measure for the similarity of atomic fluctuations of two proteins, e.g. experimental b-factors, theroetical RMSF values, or atomic fluctuations obtained from NMA.

Value

Returns the similarity coefficient(s).

Author(s)

Lars Skjaerven

References

Grant, B.J. et al. (2006) *Bioinformatics* **22**, 2695–2696. Fuglebakk, E. et al. (2013) *JCTC* **9**, 5618–5628.

See Also

Other similarity measures: covsoverlap, bhattacharyya, rmsip.

Examples

```
pdb <- read.pdb( system.file("examples/1hel.pdb", package="bio3d") )
a <- nma(pdb)
b <- nma(pdb, ff="anm")

sip(a$fluctuations, b$fluctuations)
```

---

sse.bridges *SSE Backbone Hydrogen Bonding*

Description

Determine backbone C=O to N-H hydrogen bonding in secondary structure elements.

Usage

```
sse.bridges(sse, type="helix", hbond=TRUE, energy.cut=-1.0)
```

Arguments

sse	an sse object as obtained with dssp.
type	character string specifying 'helix' or 'sheet'.
hbond	use hbond records in the dssp output.
energy.cut	cutoff for the dssp hbond energy.

Details

Simple functionality to parse the 'BP' and 'hbond' records of the DSSP output.

Requires input from function dssp with arguments resno=FALSE and full=TRUE.

Value

Returns a numeric matrix of two columns containing the residue ids of the paired residues.

Author(s)

Lars Skjaerven

References

Grant, B.J. et al. (2006) *Bioinformatics* **22**, 2695–2696.

See Also

read.pdb, dssp

Examples

```
## Not run:
# Read a PDB file
pdb <- read.pdb( system.file("examples/1hel.pdb", package="bio3d") )
sse <- dssp(pdb, resno=FALSE, full=TRUE)

sse.bridges(sse, type="helix")

## End(Not run)
```

store.atom *Store all-atom data from a PDB object*

Description

Not intended for public usage

Usage

```
store.atom(pdb)
```

Arguments

pdb	A pdb object as obtained from read.pdb

Details

This function was requested by a user and has not been extensively tested. Hence it is not yet recommended for public usage.

Value

Returns a matrix of all-atom data

Note

This function is still in development and is NOT part of the offical bio3d package

Author(s)

Barry Grant

References

Grant, B.J. et al. (2006) *Bioinformatics* **22**, 2695–2696.

See Also

read.fasta.pdb

Examples

```
## Not run:
pdb <- read.pdb( get.pdb("5p21", URLonly=TRUE) )
a <- store.atom(pdb)
a[,,1:2]

## End(Not run)
```

struct.aln *Structure Alignment Of Two PDB Files*

Description

Performs a sequence and structural alignment of two PDB entities.

Usage

```
struct.aln(fixed, mobile, fixed.inds=NULL, mobile.inds=NULL,
           write.pdbs=TRUE, outpath = "fitlsq", prefix=c("fixed",
           "mobile"), max.cycles=10, cutoff=0.5, ... )
```

Arguments

fixed	an object of class pdb as obtained from function read.pdb.
mobile	an object of class pdb as obtained from function read.pdb.
fixed.inds	atom and xyz coordinate indices obtained from atom.select that selects the elements of fixed upon which the calculation should be based.
mobile.inds	atom and xyz coordinate indices obtained from atom.select that selects the elements of mobile upon which the calculation should be based.
write.pdbs	logical, if TRUE the aligned structures are written to PDB files.
outpath	character string specifing the output directory when write.pdbs is TRUE.
prefix	a character vector of length 2 containing the filename prefix in which the fitted structures should be written.
max.cycles	maximum number of refinement cycles.
cutoff	standard deviation of the pairwise distances for aligned residues at which the fitting refinement stops.
...	extra arguments passed to seqaln function.

Details

This function performs a sequence alignment followed by a structural alignment of the two PDB entities. Cycles of refinement steps of the structural alignment are performed to improve the fit by removing atoms with a high structural deviation. The primary purpose of the function is to allow rapid structural alignment (and RMSD analysis) for protein structures with unequal, but related sequences.

The function reports the residues of fixed and mobile included in the final structural alignment, as well as the related RMSD values.

This function makes use of the underlying functions seqaln, rot.lsq, and rmsd.

Value

Returns a list with the following components:

a.inds	atom and xyz indices of fixed.
b.inds	atom and xyz indices of mobile.
xyz	fitted xyz coordinates of mobile.
rmsd	a numeric vector of RMSD values after each cycle of refinement.

Author(s)

Lars Skjarven

References

Grant, B.J. et al. (2006) *Bioinformatics* **22**, 2695–2696.

See Also

rmsd, rot.lsq, seqaln, pdbaln

Examples

```
## NOTE: FOLLOWING EXAMPLE NEEDS MUSCLE INSTALLED
if(check.utility("muscle")) {

     ## Stucture of PKA:
     a <- read.pdb("1cmk")

     ## Stucture of PKB:
     b <- read.pdb("2jdo")

     ## Align and fit b on to a:
     path = file.path(tempdir(), "struct.aln")
     aln <- struct.aln(a, b, outpath = path, outfile = tempfile())

     ## Should be the same as aln$rmsd (when using aln$a.inds and aln$b.inds)
     rmsd(a$xyz, b$xyz, aln$a.inds$xyz, aln$b.inds$xyz, fit=TRUE)

    invisible( cat("\nSee the output files:", list.files(path, full.names = TRUE), sep="\n") )
}


## Not run:
     ## Align two subunits of GroEL (open and closed states)
     a <- read.pdb("1sx4")
     b <- read.pdb("1xck")

     ## Select chain A only
     a.inds <- atom.select(a, chain="A")
```

```
b.inds <- atom.select(b, chain="A")

## Align and fit:
aln <- struct.aln(a,b, a.inds, b.inds)

## End(Not run)
```

`torsion.pdb` *Calculate Mainchain and Sidechain Torsion/Dihedral Angles*

Description

Calculate all torsion angles for a given protein PDB structure object.

Usage

```
torsion.pdb(pdb)
```

Arguments

`pdb`	a PDB structure object as obtained from function `read.pdb`.

Details

The conformation of a polypeptide chain can be usefully described in terms of angles of internal rotation around its constituent bonds. See the related `torsion.xyz` function, which is called by this function, for details.

Value

Returns a list object with the following components:

`phi`	main chain torsion angle for atoms C,N,CA,C.
`psi`	main chain torsion angle for atoms N,CA,C,N.
`omega`	main chain torsion angle for atoms CA,C,N,CA.
`alpha`	virtual torsion angle between consecutive C-alpha atoms.
`chi1`	side chain torsion angle for atoms N,CA,CB,*G.
`chi2`	side chain torsion angle for atoms CA,CB,*G,*D.
`chi3`	side chain torsion angle for atoms CB,*G,*D,*E.
`chi4`	side chain torsion angle for atoms *G,*D,*E,*Z.
`chi5`	side chain torsion angle for atoms *D,*E,*Z, NH1.
`coords`	numeric matrix of 'justified' coordinates.
`tbl`	a numeric matrix of psi, phi and chi torsion angles.

Note

For the protein backbone, or main-chain atoms, the partial double-bond character of the peptide bond between 'C=N' atoms severely restricts internal rotations. In contrast, internal rotations around the single bonds between 'N-CA' and 'CA-C' are only restricted by potential steric collisions. Thus, to a good approximation, the backbone conformation of each residue in a given polypeptide chain can be characterised by the two angles phi and psi.

Sidechain conformations can also be described by angles of internal rotation denoted chi1 up to chi5 moving out along the sidechain.

Author(s)

Barry Grant

References

Grant, B.J. et al. (2006) *Bioinformatics* **22**, 2695–2696.

See Also

torsion.xyz, read.pdb, dssp, stride.

Examples

```
##-- PDB torsion analysis
pdb <- read.pdb( "1bg2" )
tor <- torsion.pdb(pdb)
head(tor$tbl)

## basic Ramachandran plot
plot(tor$phi, tor$psi)

## torsion analysis of a single coordinate vector
inds <- atom.select(pdb,"calpha")
tor.ca <- torsion.xyz(pdb$xyz[inds$xyz], atm.inc=1)


## Not run:
##-- Compare two PDBs to highlight interesting residues
aln <- read.fasta(system.file("examples/kif1a.fa",package="bio3d"))
m <- read.fasta.pdb(aln)
a <- torsion.xyz(m$xyz[1,],1)
b <- torsion.xyz(m$xyz[2,],1)
d <- wrap.tor(a-b)
plot(m$resno[1,],d, typ="h")

## End(Not run)
```

`torsion.xyz` *Calculate Torsion/Dihedral Angles*

Description

Defined from the Cartesian coordinates of four successive atoms (A-B-C-D) the torsion or dihedral angle is calculated about an axis defined by the middle pair of atoms (B-C).

Usage

```
torsion.xyz(xyz, atm.inc = 4)
```

Arguments

`xyz`	a numeric vector of Cartisean coordinates.
`atm.inc`	a numeric value indicating the number of atoms to increment by between successive torsion evaluations (see below).

Details

The conformation of a polypeptide or nucleotide chain can be usefully described in terms of angles of internal rotation around its constituent bonds.

If a system of four atoms A-B-C-D is projected onto a plane normal to bond B-C, the angle between the projection of A-B and the projection of C-D is described as the torsion angle of A and D about bond B-C.

By convention angles are measured in the range -180 to +180, rather than from 0 to 360, with positive values defined to be in the clockwise direction.

With `atm.inc=1`, torsion angles are calculated for each set of four successive atoms contained in `xyz` (i.e. moving along one atom, or three elements of `xyz`, between sucessive evaluations). With `atm.inc=4`, torsion angles are calculated for each set of four successive non-overlapping atoms contained in `xyz` (i.e. moving along four atoms, or twelve elements of `xyz`, between sucessive evaluations).

Value

A numeric vector of torsion angles.

Note

Contributions from Barry Grant.

Author(s)

Karim ElSawy

References

Grant, B.J. et al. (2006) *Bioinformatics* **22**, 2695–2696.

See Also

torsion.pdb, pca.tor, wrap.tor, read.pdb, read.dcd.

Examples

```
## Calculate torsions for cis & trans conformers
xyz <- rbind(c(0,-0.5,0,1,0,0,1,1,0,0,1.5,0),
             c(0,-0.5,0,1,0,0,1,1,0,2,1.5,0)-3)
cis.tor   <- torsion.xyz( xyz[1,] )
trans.tor <- torsion.xyz( xyz[2,] )
apply(xyz, 1, torsion.xyz)


plot(range(xyz), range(xyz), xlab="", ylab="", typ="n", axes=FALSE)
  apply(xyz, 1, function(x){
    lines(matrix(x, ncol=3, byrow=TRUE), lwd=4)
    points(matrix(x, ncol=3, byrow=TRUE), cex=2.5,
           bg="white", col="black", pch=21) } )

  text( t(apply(xyz, 1, function(x){
    apply(matrix(x, ncol=3, byrow=TRUE)[c(2,3),], 2, mean) })),
       labels=c(0,180), adj=-0.5, col="red")

##-- PDB torsion analysis
pdb <- read.pdb("1bg2")
tor <- torsion.pdb(pdb)
## basic Ramachandran plot
plot(tor$phi, tor$psi)

## torsion analysis of a single coordinate vector
inds <- atom.select(pdb,"calpha")
tor.ca <- torsion.xyz(pdb$xyz[inds$xyz], atm.inc=3)

##-- Compare two PDBs to highlight interesting residues
aln <- read.fasta(system.file("examples/kif1a.fa",package="bio3d"))
m <- read.fasta.pdb(aln)
a <- torsion.xyz(m$xyz[1,],1)
b <- torsion.xyz(m$xyz[2,],1)
## Note the periodicity of torsion angles
d <- wrap.tor(a-b)
plot(m$resno[1,],d, typ="h")
```

`trim.pdb` *Trim a PDB Object To A Subset of Atoms.*

Description

Produce a new smaller PDB object, containing a subset of atoms, from a given larger PDB object.

Usage

```
trim.pdb(pdb, inds = NULL, sse = TRUE)
```

Arguments

`pdb`	a PDB structure object obtained from `read.pdb`.
`inds`	a list object of ATOM and XYZ indices as obtained from `atom.select`.
`sse`	logical, if 'FALSE' helix and sheet components are omitted from output.

Details

This is a basic utility function for creating a new PDB object based on a selection of atoms.

Value

Returns a list of class `"pdb"` with the following components:

`atom`	a character matrix containing all atomic coordinate ATOM data, with a row per ATOM and a column per record type. See below for details of the record type naming convention (useful for accessing columns).
`het`	a character matrix containing atomic coordinate records for atoms within "non-standard" HET groups (see `atom`).
`helix`	'start', 'end' and 'length' of H type sse, where start and end are residue numbers "resno".
`sheet`	'start', 'end' and 'length' of E type sse, where start and end are residue numbers "resno".
`seqres`	sequence from SEQRES field.
`xyz`	a numeric vector of ATOM coordinate data.
`xyz.models`	a numeric matrix of ATOM coordinate data for multi-model PDB files.
`calpha`	logical vector with length equal to `nrow(atom)` with TRUE values indicating a C-alpha "elety".

Note

het and seqres list components are returned unmodified.

For both atom and het list components the column names can be used as a convenient means of data access, namely: Atom serial number "eleno", Atom type "elety", Alternate location indicator "alt", Residue name "resid", Chain identifier "chain", Residue sequence number "resno", Code for insertion of residues "insert", Orthogonal coordinates "x", Orthogonal coordinates "y", Orthogonal coordinates "z", Occupancy "o", and Temperature factor "b". See examples for further details.

Author(s)

Barry Grant, Lars Skjaerven

References

Grant, B.J. et al. (2006) *Bioinformatics* **22**, 2695–2696.

For a description of PDB format (version3.3) see:
http://www.wwpdb.org/documentation/format33/v3.3.html. .

See Also

read.pdb, atom.select, write.pdb

Examples

```
## Not run:
## Read a PDB file from the RCSB online database
pdb <- read.pdb("1bg2")

## Select calpha atoms
sele <- atom.select(pdb, "calpha")

## Trim PDB
new.pdb <- trim.pdb(pdb, inds=sele)

## Write to file
write.pdb(new.pdb, file="calpha.pdb")

## End(Not run)
```

---

unbound *Sequence Generation from a Bounds Vector*

Description

Generate a sequence of consecutive numbers from a bounds vector.

Usage

```
unbound(start, end)
```

Arguments

start	vector of starting values, such as that obtained from bounds.
end	vector of (maximal) end values, such as that obtained from bounds.

Details

This is a simple utility function that does the opposite of the bounds function.

Value

Returns a numeric sequence vector.

Author(s)

Barry Grant

References

Grant, B.J. et al. (2006) *Bioinformatics* **22**, 2695–2696.

See Also

bounds

Examples

```
test <- c(seq(1,5,1),8,seq(10,15,1))
b <- bounds(test)
unbound(b[,"start"],b[,"end"])
```

---

uniprot *Fetch UniProt Entry Data.*

Description

Fetch protein sequence and functional information from the UniProt database.

Usage

```
uniprot(accid)
```

Arguments

accid	UniProt accession id.

Details

This is a basic utility function for downloading information from the UniProt database. UniProt contains protein sequence and functional information.

Value

Returns a list object with the following components:

accession	a character vector with UniProt accession id's.
name	abbreviated name.
fullName	full recommended protein name.
shortName	short protein name.
sequence	protein sequence.
gene	gene names.
organism	organism.
taxon	taxonomic lineage.

Author(s)

Lars Skjaerven

References

Grant, B.J. et al. (2006) *Bioinformatics* **22**, 2695–2696.

See also the UniProt web-site for more information:
http://www.uniprot.org/.

See Also

blast.pdb, get.seq

Examples

```
prot <- uniprot('PH4H_HUMAN')
prot$fullName
prot$sequence
```

var.xyz *Pairwise Distance Variance in Cartesian Coordinates*

Description

Calculate the variance of all pairwise distances in an ensemble of Cartesian coordinates.

Usage

```
var.xyz(xyz, weights=TRUE)
var.pdbs(pdbs, ...)
```

Arguments

xyz	an object of class "xyz" containing Cartesian coordinates in a matrix.
weights	logical, if TRUE weights are calculated based on the pairwise distance variance.
pdbs	a 'pdbs' object as object from function pdbaln.
...	arguments passed to associated functions.

Details

This function calculates the variance of all pairwise distances in an ensemble of Cartesian coordinates. The primary use of this function is to calculate weights to scale the pair force constant for NMA.

Value

Returns the a matrix of the pairwise distance variance, formated as weights if 'weights=TRUE'.

Author(s)

Lars Skjaerven

References

Grant, B.J. et al. (2006) *Bioinformatics* **22**, 2695–2696.

See Also

nma.pdbs

vec2resno *Replicate Per-residue Vector Values*

Description

Replicate values in one vector based on consecutive entries in a second vector. Useful for adding per-residue data to all-atom PDB files.

Usage

```
vec2resno(vec, resno)
```

Arguments

vec	a vector of values to be replicated.
resno	a reference vector or a PDB structure object, obtained from read.pdb, upon which replication is based.

Details

This function can aid in mapping data to PDB structure files. For example, residue conservation per position (or any other one value per residue data) can be replicated to fit the B-factor field of an all atom PDB file which can then be rendered according to this field in a molecular viewer.

A basic check is made to ensure that the number of consecutively unique entries in the reference vector equals the length of the vector to be replicated.

Value

Returns a vector of replicated values.

Author(s)

Barry Grant

References

Grant, B.J. et al. (2006) *Bioinformatics* **22**, 2695–2696.

See Also

read.pdb, atom.select, write.pdb

Examples

```
vec2resno(c("a","b"), c(1,1,1,1,2,2))
```

view.cna *View CNA Protein Structure Network Community Output in VMD*

Description

This function generates a VMD scene file and a PDB file that can be read and rendered by the VMD molecular viewer. Chose 'color by chain' to see corresponding regions of structure colored by community along with the community protein structure network.

Usage

```
view.cna(x, pdb, layout = layout.cna(x, pdb, k=3),
col.sphere=NULL, col.lines = "silver", weights = NULL,
radius = table(x$communities$membership)/5, alpha = 1,
vmdfile = "network.vmd", pdbfile = "network.pdb",
launch = FALSE)
```

Arguments

x	A 'cna' class object such as obtained from 'cna' function/
pdb	A 'pdb' class object such as obtained from 'read.pdb' function.
layout	A numeric matrix of Nx3 XYZ coordinate matrix, where N is the number of community spheres to be drawn.
col.sphere	A numeric vector containing the sphere colors.
col.lines	A character object specifying the color of the edges (default 'silver'). Must use VMD colors names.
weights	A numeric vector specifying the edge width. Default is taken from E(x$community.network)$weight.
radius	A numeric vector containing the sphere radii. Default is taken from the number of community members divided by 5.
alpha	A single element numeric vector specifying the VMD alpha transparency parameter. Default is set to 1.
vmdfile	A character element specifying the output VMD scene file name that will be loaded in VMD.
pdbfile	A character element specifying the output pdb file name to be loaded in VMD.
launch	Logical. If TRUE, a VMD session will be started with the output of 'view.cna'.

Details

This function generates a scaled sphere (communities) and stick (edges) representation of the community network along with the corresponding protein structure divided into chains, one chain for each community. The sphere radii are proportional to the number of community members and the edge widths correspond to network edge weights.

Value

Two files are generated as output. A pdb file with the residue chains assigned according to the community and a text file containing The drawing commands for the community representation.

Author(s)

Barry Grant

References

Humphrey, W., Dalke, A. and Schulten, K., "VMD - Visual Molecular Dynamics" J. Molec. Graphics 1996, 14.1, 33-38.

Examples

```
## Not run:

# Load the correlation network from MD data
attach(hivp)

# Read the starting PDB file to determine atom correspondence
pdbfile <- system.file("examples/hivp.pdb", package="bio3d")
pdb <- read.pdb(pdbfile)

# View cna
view.cna(net, pdb, launch=FALSE)
## within VMD set 'coloring method' to 'Chain' and 'Drawing method' to Tube


##-- From NMA
pdb.gdi = read.pdb("1KJY")
pdb.gdi = trim.pdb(pdb.gdi, inds=atom.select(pdb.gdi, chain="A", elety="CA"))
modes.gdi = nma(pdb.gdi)
cij.gdi = dccm(modes.gdi)
net.gdi = cna(cij.gdi, cutoff.cij=0.35)
#view.cna(net.gdi, pdb.gdi, alpha = 0.7, launch=TRUE)

## End(Not run)
```

---

view.dccm *Visualization of Dynamic Cross-Correlation*

Description

Structural visualization of a cross-correlation matrix.

Usage

```
view.dccm(dccm, pdb, step=0.2, omit=0.2, radius=0.15, type="pymol", outprefix="corr",
          launch=FALSE, exefile="pymol")
```

Arguments

dccm	an object of class dccm as obtained from function dccm or dccm.nma.
pdb	an object of class pdb as obtained from function read.pdb or a numerical vector of Cartesian coordinates.
step	binning interval of cross-correlation coefficents.
omit	correlation coefficents with values (0-omit, 0+omit) will be omitted from visualization.
radius	radius of visualized correlations.
type	character string specifying the type of visualization: 'pymol' or 'pdb'.
outprefix	character string specifying the file prefix. If NULL the temp directory will be used.
launch	logical, if TRUE PyMol will be launched.
exefile	file path to the 'PYMOL' program on your system (i.e. how is 'PYMOL' invoked).

Details

This function generates a PyMOL (python) script that will draw colored lines between (anti)correlated residues. The PyMOL script file is stored in the working directory with filename "corr.py", with coordinates in PDB format with filename "corr.inpcrd.pdb". PyMOL will only be launched when using argument 'launch=TRUE'. Alternatively a PDB file with CONECT records will be generated (when argument type='pdb').

For the PyMOL version, PyMOL CGO objects are generated - each object representing a range of correlation values (corresponding to the actual correlation values as found in the correlation matrix). E.g. the PyMOL object with name "cor_-1_-08" would display all pairs of correlations with values between -1 and -0.8.

Value

Called for its action.

Author(s)

Lars Skjaerven

References

Grant, B.J. et al. (2006) *Bioinformatics* **22**, 2695–2696.

See Also

nma, dccm

Examples

```
## Not run:
## Fetch stucture
pdb <- read.pdb( system.file("examples/1hel.pdb", package="bio3d") )

## Calculate normal modes
modes <- nma(pdb)

## Calculate correlation matrix
cm <- dccm.nma(modes)

view.dccm(cm, modes$xyz)

## End(Not run)
```

---

view.modes *Vector Field Visualization of Modes*

Description

Structural visualization of mode vectors obtained from PCA or NMA.

Usage

```
view.modes(modes, mode=NULL, outprefix="mode_vecs",
          scale=5, dual=FALSE, launch=FALSE, exefile="pymol")
```

Arguments

modes	an object of class nma or pca as obtained from functions nma or pca.xyz.
mode	the mode number for which the vector field should be made.
outprefix	character string specifying the file prefix. If NULL the temp directory will be used.
scale	global scaling factor.
dual	logical, if TRUE mode vectors are also drawn in both direction.
launch	logical, if TRUE PyMol will be launched.
exefile	file path to the 'PYMOL' program on your system (i.e. how is 'PYMOL' invoked).

Details

This function generates a PyMOL (python) script for drawing mode vectors on a PDB structure. The PyMOL script file is stored in the working directory with filename "mode_vecs.py", with coordinates in PDB format "mode_vecs.inpcrd.pdb". PyMOL will only be launched when using argument 'launch=TRUE'.

Value

Called for its action.

Author(s)

Lars Skjaerven

References

Grant, B.J. et al. (2006) *Bioinformatics* **22**, 2695–2696.

See Also

nma, pca.xyz

Examples

```
## Not run:
## Fetch stucture
pdb <- read.pdb( system.file("examples/1hel.pdb", package="bio3d") )

## Calculate normal modes
modes <- nma(pdb)

view.modes(modes, mode=7)

## End(Not run)
```

---

vmd.colors *VMD Color Palette*

Description

This function creates a character vector of the colors used by the VMD molecular graphics program.

Usage

```
vmd.colors(n=33, picker=FALSE, ...)
```

Arguments

n	The number of desired colors chosen in sequence from the VMD color palette (>=1)
picker	Logical, if TRUE a color wheel plot will be produced to aid with color choice.
...	Extra arguments passed to the rgb function, including alpha transparency.

Details

The function uses the underlying 33 RGB color codes from VMD, See http://www.ks.uiuc.edu/Research/vmd/. Note that colors will be recycled if "n" > 33 with a warning issued. When 'picker' is set to "TRUE" a color wheel of the requested colors will be plotted to the currently active device.

Value

Returns a character vector with color names.

Author(s)

Barry Grant

References

Grant, B.J. et al. (2006) *Bioinformatics* **22**, 2695–2696.

http://www.ks.uiuc.edu/Research/vmd/

See Also

bwr.colors

Examples

```
## Generate a vector of 10 colors
clrs <- vmd.colors(10)
vmd.colors(4, picker=TRUE)
```

---

wrap.tor *Wrap Torsion Angle Data*

Description

Adjust angular data so that the absolute difference of any of the observations from its mean is not greater than 180 degrees.

Usage

```
wrap.tor(data, wrapav=TRUE, avestruc=NULL)
```

Arguments

data	a numeric vector or matrix of torsion angle data as obtained from torsion.xyz.
wrapav	logical, if TRUE average structure is also 'wrapped'
avestruc	a numeric vector corresponding to the average structure

Details

This is a basic utility function for coping with the periodicity of torsion angle data, by 'wraping' angular data such that the absolute difference of any of the observations from its column-wise mean is not greater than 180 degrees.

Value

A numeric vector or matrix of wrapped torsion angle data.

Author(s)

Karim ElSawy

References

Grant, B.J. et al. (2006) *Bioinformatics* **22**, 2695–2696.

See Also

`torsion.xyz`

---

write.crd *Write CRD File*

Description

Write a CHARMM CARD (CRD) coordinate file.

Usage

```
write.crd(pdb = NULL, xyz = pdb$xyz, resno = NULL, resid = NULL,
   eleno = NULL, elety = NULL, segid = NULL, resno2 = NULL, b = NULL,
   verbose = FALSE, file = "R.crd")
```

Arguments

pdb	a structure object obtained from `read.pdb` or `read.crd`.
xyz	Cartesian coordinates as a vector or 3xN matrix.
resno	vector of residue numbers of length equal to length(xyz)/3.
resid	vector of residue types/ids of length equal to length(xyz)/3.
eleno	vector of element/atom numbers of length equal to length(xyz)/3.
elety	vector of element/atom types of length equal to length(xyz)/3.
segid	vector of segment identifiers with length equal to length(xyz)/3.
resno2	vector of alternate residue numbers of length equal to length(xyz)/3.
b	vector of weighting factors of length equal to length(xyz)/3.
verbose	logical, if TRUE progress details are printed.
file	the output file name.

Details

Only the xyz argument is strictly required. Other arguments assume a default poly-ALA C-alpha structure with a blank segid and B-factors equal to 0.00.

Value

Called for its effect.

Note

Check that resno and eleno do not exceed "9999".

Author(s)

Barry Grant

References

Grant, B.J. et al. (2006) *Bioinformatics* **22**, 2695–2696.

For a description of CHARMM CARD (CRD) format see:
http://www.charmmtutorial.org/index.php/CHARMM:The_Basics.

See Also

read.crd, read.pdb, atom.select, write.pdb, read.dcd, read.fasta.pdb, read.fasta

Examples

```
## Not run:
# Read a PDB file
pdb <- read.pdb( "1bg2" )
summary(pdb)
# Convert to CHARMM format
new <- convert.pdb(pdb, type="charmm")
summary(new)
# Write a CRD file
write.crd(new, file="4charmm.crd")

## End(Not run)
```

---

write.fasta *Write FASTA Formated Sequences*

Description

Write aligned or un-aligned sequences to a FASTA format file.

Usage

```
write.fasta(alignment=NULL, ids=NULL, seqs=alignment$ali, file, append = FALSE)
```

Arguments

alignment	an alignment list object with id and ali components, similar to that generated by read.fasta.
ids	a vector of sequence names to serve as sequence identifers
seqs	an sequence or alignment character matrix or vector with a row per sequence
file	name of output file.
append	logical, if TRUE output will be appended to file; otherwise, it will overwrite the contents of file.

Value

Called for its effect.

Note

For a description of FASTA format see: http://www.ncbi.nlm.nih.gov/BLAST/blastcgihelp.shtml.

Author(s)

Barry Grant

References

Grant, B.J. et al. (2006) *Bioinformatics* **22**, 2695–2696.

See Also

read.fasta, read.fasta.pdb

Examples

```
## Read a PDB file
pdb <- read.pdb("1bg2")

## Extract sequence from PDB file
s <- aa321(pdb$seqres)                    # SEQRES
a <- aa321(pdb$atom[pdb$calpha,"resid"]) # ATOM

## Write simple fasta file
#write.fasta( seqs=seqbind(s,a), file="eg.fa")
#write.fasta( ids=c("seqres","atom"), seqs=seqbind(s,a), file="eg.fa" )

outfile1 = file.path(tempdir(), "eg.fa")
```

```
write.fasta(list( id=c("seqres"),ali=s ), file = outfile1)
write.fasta(list( id=c("atom"),ali=a ), file = outfile1, append=TRUE)

## Align seqres and atom records
#seqaln(seqbind(s,a))

## Read alignment
aln<-read.fasta(system.file("examples/kif1a.fa",package="bio3d"))

## Cut all but positions 130 to 245
aln$ali=aln$ali[,130:245]

outfile2 = file.path(tempdir(), "eg2.fa")
write.fasta(aln, file = outfile2)

invisible( cat("\nSee the output files:", outfile1, outfile2, sep="\n") )
```

`write.ncdf` *Write AMBER Binary netCDF files*

Description

Write coordinate data to a binary netCDF trajectory file.

Usage

```
write.ncdf(x, trjfile = "R.ncdf", cell = NULL)
```

Arguments

`x`	A numeric matrix of xyz coordinates with a frame/structure per row and a Cartesian coordinate per column.
`trjfile`	name of the output trajectory file.
`cell`	A numeric matrix of cell information with a frame/structure per row and a cell length or angle per column. If NULL cell will not be written.

Details

Writes an AMBER netCDF (Network Common Data Form) format trajectory file with the help of David W. Pierce's (UCSD) ncdf package available from CRAN.

Value

Called for its effect.

Note

See AMBER documentation for netCDF format description.

NetCDF binary trajectory files are supported by the AMBER modules sander, pmemd and ptraj. Compared to formatted trajectory files, the binary trajectory files are smaller, higher precision and significantly faster to read and write.

NetCDF provides for file portability across architectures, allows for backwards compatible extensibility of the format and enables the files to be self-describing. Support for this format is available in VMD.

Author(s)

Barry Grant

References

Grant, B.J. et al. (2006) *Bioinformatics* **22**, 2695–2696. http://www.unidata.ucar.edu/packages/netcdf/ http://cirrus.ucsd.edu/~pierce/ncdf/ http://ambermd.org/netcdf/nctraj.html

See Also

read.dcd, read.ncdf, read.pdb, write.pdb, atom.select

Examples

```
## Not run:
##-- Read example trajectory file
trtfile <- system.file("examples/hivp.dcd", package="bio3d")
trj <- read.dcd(trtfile)

## Write to netCDF format
write.ncdf(trj, "newtrj.nc")

## Read trj
trj <- read.ncdf("newtrj.nc")

## End(Not run)
```

---

write.pdb *Write PDB Format Coordinate File*

Description

Write a Protein Data Bank (PDB) file for a given 'xyz' Cartesian coordinate vector or matrix.

Usage

```
write.pdb(pdb = NULL, file = "R.pdb", xyz = pdb$xyz, type = NULL, resno = NULL,
 resid = NULL, eleno = NULL, elety = NULL, chain = NULL, insert = NULL,
 alt = NULL, o = NULL, b = NULL, segid = NULL, elesy = NULL, charge = NULL,
append = FALSE, verbose = FALSE, chainter = FALSE, end = TRUE, print.segid = FALSE)
```

Arguments

pdb	a PDB structure object obtained from read.pdb.
file	the output file name.
xyz	Cartesian coordinates as a vector or 3xN matrix.
type	vector of record types, i.e. "ATOM" or "HETATM", with length equal to length(xyz)/3.
resno	vector of residue numbers of length equal to length(xyz)/3.
resid	vector of residue types/ids of length equal to length(xyz)/3.
eleno	vector of element/atom numbers of length equal to length(xyz)/3.
elety	vector of element/atom types of length equal to length(xyz)/3.
chain	vector of chain identifiers with length equal to length(xyz)/3.
insert	vector of insertion code with length equal to length(xyz)/3.
alt	vector of alternate record with length equal to length(xyz)/3.
o	vector of occupancy values of length equal to length(xyz)/3.
b	vector of B-factors of length equal to length(xyz)/3.
segid	vector of segment id of length equal to length(xyz)/3.
elesy	vector of element symbol of length equal to length(xyz)/3.
charge	vector of atomic charge of length equal to length(xyz)/3.
append	logical, if TRUE output is appended to the bottom of an existing file (used primarly for writing multi-model files).
verbose	logical, if TRUE progress details are printed.
chainter	logical, if TRUE a TER line is inserted at termination of a chain.
end	logical, if TRUE END line is written.
print.segid	logical, if FALSE segid will not be written.

Details

Only the xyz argument is strictly required. Other arguments assume a default poly-ALA C-alpha structure with a blank chain id, occupancy values of 1.00 and B-factors equal to 0.00.

If the input argument xyz is a matrix then each row is assumed to be a different structure/frame to be written to a "multimodel" PDB file, with frames separated by "END" records.

Value

Called for its effect.

Note

Check that: (1) `chain` is one character long e.g. "A", and (2) `resno` and `eleno` do not exceed "9999".

Author(s)

Barry Grant with contributions from Joao Martins.

References

Grant, B.J. et al. (2006) *Bioinformatics* **22**, 2695–2696.

For a description of PDB format (version3.3) see:
http://www.wwpdb.org/documentation/format33/v3.3.html.

See Also

`read.pdb`, `read.dcd`, `read.fasta.pdb`, `read.fasta`

Examples

```
# Read a PDB file
pdb <- read.pdb( "1bg2" )

# Renumber residues
nums <- as.numeric(pdb$atom[,"resno"])
nums <- nums - (nums[1] - 1)

# Write out renumbered PDB file
outfile = file.path(tempdir(), "eg.pdb")
write.pdb(pdb=pdb, resno = nums, file = outfile)

invisible( cat("\nSee the output file:", outfile, sep = "\n") )
```

---

write.pir *Write PIR Formated Sequences*

Description

Write aligned or un-aligned sequences to a PIR format file.

Usage

```
write.pir(alignment=NULL, ids=NULL, seqs=alignment$ali,
   pdb.file = NULL, chain.first = NULL, resno.first = NULL,
   chain.last = NULL, resno.last = NULL, file, append = FALSE)
```

Arguments

alignment	an alignment list object with id and ali components, similar to that generated by read.fasta.
ids	a vector of sequence names to serve as sequence identifers
seqs	an sequence or alignment character matrix or vector with a row per sequence
pdb.file	a vector of pdb filenames; For sequence, provide "".
chain.first	a vector of chain id for the first residue.
resno.first	a vector of residue number for the first residue.
chain.last	a vector of chain id for the last residue.
resno.last	a vector of residue number for the last residue.
file	name of output file.
append	logical, if TRUE output will be appended to file; otherwise, it will overwrite the contents of file.

Value

Called for its effect.

Note

PIR is required format for input alignment file to use Modeller. For a description of PIR format see: https://salilab.org/modeller/manual/node488.html.

Author(s)

Xin-Qiu Yao

References

Grant, B.J. et al. (2006) *Bioinformatics* **22**, 2695–2696.

See Also

read.fasta, read.fasta.pdb, write.fasta

Examples

```
## NOTE: FOLLOWING EXAMPLE NEEDS MUSCLE INSTALLED
if(check.utility("muscle")) {

## Generate an input file for structural modeling of
## transducin G-alpha subunit using the template 3SN6_A

## Read transducin alpha subunit sequence
seq <- get.seq("P04695", db = "uniprot", outfile = tempfile())
```

```
## Read structure template
path = tempdir()
pdb.file <- get.pdb("3sn6_A", path = path, split = TRUE)
pdb <- read.pdb(pdb.file)

## Build an alignment between template and target
aln <- seqaln(seqbind(pdbseq(pdb), seq), id = c("3sn6_A", seq$id), outfile = tempfile())

## Write PIR format alignment file
outfile = file.path(tempdir(), "eg.pir")
write.pir(aln, pdb.file = c(pdb.file, ""), file = outfile)

invisible( cat("\nSee the output file:", outfile, sep = "\n") )

}
```

`write.pqr` *Write PQR Format Coordinate File*

Description

Write a PQR file for a given 'xyz' Cartesian coordinate vector or matrix.

Usage

```
write.pqr(pdb = NULL, xyz = pdb$xyz, resno = NULL, resid = NULL, eleno =
NULL, elety = NULL, chain = NULL, o = NULL, b = NULL, het = FALSE,
append = FALSE, verbose = FALSE, chainter = FALSE, file = "R.pdb")
```

Arguments

pdb	a PDB structure object obtained from read.pdb.
xyz	Cartesian coordinates as a vector or 3xN matrix.
resno	vector of residue numbers of length equal to length(xyz)/3.
resid	vector of residue types/ids of length equal to length(xyz)/3.
eleno	vector of element/atom numbers of length equal to length(xyz)/3.
elety	vector of element/atom types of length equal to length(xyz)/3.
chain	vector of chain identifiers with length equal to length(xyz)/3.
o	vector of occupancy values of length equal to length(xyz)/3.
b	vector of B-factors of length equal to length(xyz)/3.
het	logical, if TRUE 'HETATM' records from pdb object are written to the output PDB file.
append	logical, if TRUE output is appended to the bottom of an existing file (used primarly for writing multi-model files).

verbose	logical, if TRUE progress details are printed.
chainter	logical, if TRUE a TER line is inserted between chains.
file	the output file name.

Details

Only the xyz argument is strictly required. Other arguments assume a default poly-ALA C-alpha structure with a blank chain id, occupancy values of 1.00 and B-factors equal to 0.00.

If the input argument xyz is a matrix then each row is assumed to be a different structure/frame to be written to a "multimodel" PDB file, with frames separated by "END" records.

Value

Called for its effect.

Note

Check that: (1) chain is one character long e.g. "A", and (2) resno and eleno do not exceed "9999".

Author(s)

Barry Grant with contributions from Joao Martins.

References

Grant, B.J. et al. (2006) *Bioinformatics* **22**, 2695–2696.

For a description of PDB format (version3.3) see:
http://www.wwpdb.org/documentation/format33/v3.3.html.

See Also

read.pdb, read.dcd, read.fasta.pdb, read.fasta

Examples

```
# Read a PDB file
pdb <- read.pdb( "1bg2" )

# Renumber residues
nums <- as.numeric(pdb$atom[,"resno"])
nums <- nums - (nums[1] - 1)

# Write out renumbered PDB file
outfile = file.path(tempdir(), "eg.pdb")
write.pdb(pdb=pdb, resno = nums, file = outfile)

invisible( cat("\nSee the output file:", outfile, sep = "\n") )
```

Index