The Bio3D Package

November 13, 2013

Title Biological Structure Analysis

Version 2.0-1

Author Barry Grant, Xin-Qiu Yao, Lars Skjaerven

VignetteBuilder knitr

Suggests ncdf, lattice, grid, bigmemory, multicore, XML, knitr

Depends R (>= 3.0.0)

LazyData yes

Description The Bio3D package contains 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 database searches, atom summaries, atom selection, re-orientation, superposition, rigid core identification, clustering, torsion analysis, distance matrix analysis, structure and sequence conservation analysis, and principal component analysis (PCA). 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 URL below for more information.

Maintainer Barry Grant `<bjgrant@umich.edu>`

License GPL (>= 2)

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

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:	2.0-1
Date: 2013-11-13	
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 database searches (`blast.pdb`), atom summaries (`summary.pdb`), atom selection (`atom.select`), re-orientation (`orient.pdb`), superposition (`rot.lsq`, `fit.xyz`), rigid core identification (`core.find`, `plot.core`, `fit.xyz`), torsion/dihedral analysis (`torsion.pdb`, `torsion.xyz`), clustering (via `hclust`), principal component analysis (`pca.xyz`, `pca.tor`, `plot.pca`, `plot.pca.loadings`, `mktrj.pca`) and dynamical cross-correlation analysis (`dccm`, `plot.dccm`) 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 (elastic network model) with (`nma`) and (`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` and `pairwise`).

Note

The latest version and further documentation 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>

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:
##-- Read a PDB file
pdb <- read.pdb("1BG2")

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

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

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

## write a FASTA format sequence file
write.fasta(seqs=seqbind(s, 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])

## 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])

#a <- mktrj.pca(pc.xray, pc=1, resno =rn, resid = rd, file="pc1.pdb")
#b <- mktrj.pca(pc.xray, pc=2, resno =rn, resid = rd, file="pc2.pdb")
```

```
c <- 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,"///23:31,84:87///CA/")
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...
```

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)
```

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 <- atom2mass(pdb$atom[1:10,"resid"])

## Dealing with unconventional residues
pdb <- read.pdb("1xj0", het2atom=TRUE)

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

```
## 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="eg.html")
aln2html(aln, colorscheme="ent", file="eg.html")
## View/open the file <a href='eg.html'>"eg.html"</a> in your web browser
```

`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 *Index of Atomic Masses*

Description

A dictonary of atomic masses and PDB atom names to atom elements mapping.

Usage

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

Format

A list with the following components:

1. elety list: PDB atom name indices and associated atom element types.
2. mass list: relative atomic masses.

Source

Most text books in chemistry.
See also: http://en.wikipedia.org/wiki/Relative_atomic_mass

References

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

Examples

```
## Load atom index data
data(atom.index)

# Examine the masses
atom.index$mass

# Fetch mass for an atom element
atom.index$mass[["C"]]

# Examine atom name to atom element mappings
atom.index$elety

# Fetch atom element for a specific PDB atom name
atom.index$elety[["SD"]]
```

`atom.select` *Atom Selection From PDB Structure*

Description

Return the atom and xyz coordinates 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, verbose=TRUE, rm.insert=FALSE)
```

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.
`verbose`	logical, if TRUE details of the selection are printed.
`rm.insert`	logical, if TRUE insert ATOM records from the `pdb` object are ignored.

Details

This function allows for the selection of atom and coordinate data corresponding to the intersection of input criteria (such as a 'chain', 'resno', 'elety' etc. Or a hierarchical 'selection string' `string` with a strict format.

The selection `string` should be a single element character vector containing a string composed of six sections separated by a '/' character.

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 `chain="A", resno=65:143, elety="CA"`. In addition, the character `string` shortcuts `"calpha"`, `"back"`, `"backbone"`, `"cbeta"`, `"h"` and `"noh"` may also be used. See below for examples.

When called without a selection string, `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

read.pdb, write.pdb, read.dcd

Examples

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

# Print structure summary
atom.select(pdb)

# Select all C-alpha atoms with residues numbers between 65 and 143
ca.inds <- atom.select(pdb, resno=65:143, elety="CA")
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, "///65:143///CA/")
inds <- atom.select(pdb, "///130:142///N,CA,C,O/")

## End(Not run)
```

---

atom2mass *Atom Names to Mass Converter*

Description

Convert atom names to atomic mass.

Usage

```
atom2mass(pdb, inds=NULL, mass.custom=NULL, elety.custom=NULL,
          grpby=NULL, rescue=TRUE)

atom2ele(pdb, inds=NULL, elety.custom=NULL, rescue=TRUE)

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

Arguments

`pdb`	a character vector containing the atom names to convert to atomic masses. Alternatively, an 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 (in effect only when a `pdb` object is provided).
`mass.custom`	a list of atom elements and their corresponding masses.
`elety.custom`	a list of atom names to element symbol mapping.
`grpby`	a vector counting connective duplicated elements that indicate the elements of `atoms` that should be considered as a group (e.g. atoms from a particular residue).
`rescue`	logical, if TRUE the atom element will be mapped to the first character of the atom name.
`form`	character string containing the chemical formula at which the mass calculation should be based.
`sum.mass`	logical, if TRUE the sum of masses is returned.

Details

This function converts atom names to their corresponding relative atomic masses. Atom names found in standard amino acids in the PDB are mapped to atom elements (with `atom2ele`) which finally are mapped to mass.

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.

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.

See examples for more details.

Value

Returns a numeric vector of atomic masses.

Author(s)

Lars Skjaerven

References

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

See Also

atom.index, com

Examples

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

formula2mass("C5 H6 N O3")

## Not run:
## Fetch atomic masses in a PDB object
pdb <- read.pdb("1hel")
masses <- atom2mass(pdb)

## or
masses <- atom2mass(pdb$atom[1:10,"elety"])

## Group and sum per residue
masses <- atom2mass(pdb, grpby=pdb$atom[,"resno"])


## Map atom names manually
pdb <- read.pdb("3RE0", het2atom=TRUE)
inds <- atom.select(pdb, resno=201)

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

masses <- atom2mass(pdb, inds, mass.custom=mass.cust, elety.custom=elety.cust)

## End(Not run)
```

---

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 )
```

`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', het2atom=TRUE)

     # 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")
```

Arguments

`seq`	a single element or multi-element character vector containing the query sequence.
`database`	a single element character vector specifying the database against which to search. Current options are 'pdb', 'nr' and 'swissprot'.

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 seven 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.

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

`seqaln`

Examples

```
pdb <- read.pdb("1bg2")
blast <- blast.pdb( pdbseq(pdb) )

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

---

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)
```

`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 multicore package 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)
```

`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", het2atom=TRUE)
inds <- atom.select(pdb, resid="LL2")
mycom <- com(pdb, inds, rescue=TRUE)
warnings()


## Map atom names manually
pdb <- read.pdb("3RE0", het2atom=TRUE)
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( "http://www.rcsb.org/pdb/files/1BG2.pdb" )

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

# Select all atoms with residues numbers between 65 and 143
res.inds <- atom.select(pdb, resno=65:143)

# Select all C-beta atoms with residues numbers between 65 and 143
cb.inds <- atom.select(pdb, resno=65:143, 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="-")
```

`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` *Convert Between Various PDB formats*

Description

Convert between CHARMM, Amber, Gromacs and Brookhaven PDB formats.

Usage

```
convert.pdb(pdb, type, renumber = FALSE, first.resno = 1, first.eleno = 1, rm.h
```

Arguments

`pdb`	a structure object of class `"pdb"`, obtained from `read.pdb`.
`type`	output format.
`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.
`rm.h`	logical, if TRUE hydrogen atoms are removed.
`rm.wat`	logical, if TRUE water atoms are removed.

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( "1bg2" )
summary(pdb)
# Convert to CHARMM format
new <- convert.pdb(pdb, type="charmm")
summary(new)
# Write a PDB and CRD file
#write.pdb(new, file="4charmm.pdb")
#write.crd(new, file="4charmm.crd")
```

`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 multicore package 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.
`atom`	atom indices of core atoms at each round.
`xyz`	xyz indices of core atoms at each 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)
```

---

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.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.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`, `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, ...)
```

Arguments

`x`	an object of class `enma` as obtained from function `nma.pdbs`.
`...`	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

```
## 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)

## Normal mode analysis on aligned data
all.modes <- nma.pdbs(pdbs, full=TRUE, fit=TRUE, rm.gaps=TRUE)

## Calculate correlation matrices
all.dccm <- dccm(all.modes)
```

`dccm.mean` *Atomic Correlation Matrix Consensus Filtering*

Description

Filter a three-dimensional correlation matrix, or DCCM, composed of results from multiple dccm calculations and return correlations present in at least a chosen subset of the calculated matrices.

Usage

```
## S3 method for class 'mean'
dccm(x, cutoff.sims = dim(x)[3], cutoff.cij = 0.4, ...)
```

Arguments

`x`	A numeric array with 3 dimensions '(NxNxZ)' containing atomic correlation values for N residues or atoms from Z calculations.
`cutoff.sims`	A single element numeric vector corresponding to the minimum number of Z dimensions (i.e. dccm matrices) a correlation between two residues must be present and whose absolute value must be higher than `cutoff.cij`. If not, the matrix element will be set to 0 and not used for the average calculation.
`cutoff.cij`	A single element numeric vector corresponding to the minimum absolute value a correlation element must be equal to or higher to be used for the average calculation. If lower, the element will be set to 0 and not used for the average calculation.
`...`	extra arguments that are currently ignored.

Value

Returns a numeric 'NxN' atomic correlation matrix of class 'dccm'.

Author(s)

Guido Scarabelli & Barry Grant

References

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

See Also

dccm, dccm.nma, dccm.xyz, plot.dccm

Examples

```
## Not run:
# 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
raw.files <- get.pdb(ids, path = "raw_pdbs")
files <- pdbsplit(raw.files, ids)

# 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 <- dccm.mean(cij$all.dccm, cutoff.sims = 8, cutoff.cij = 0)
plot.dccm(cij.all, main = "Consensus Residue Cross Correlation")

## End(Not run)
```

`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 multicore package 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("1hel")

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

Description

Determine the cross-correlations of atomic displacements.

Usage

```
## S3 method for class 'xyz'
dccm(x, reference = apply(xyz, 2, mean), grpby=NULL,
ncore=1, nseg.scale=1, ...)
```

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

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.

Value

Returns a cross-correlation matrix.

Note

This function is currently very basic i.e. inefficient and **SLOW**.

Author(s)

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.

See Also

`cor` for examining xyz cross-correlations, `pca.xyz`.

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 multicore package 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("1hel")

## 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 multicore package 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 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)


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

# 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.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)
```

`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 amino-acids, 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/transducin data. 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 input and processing before execution.

Usage

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

Format

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

1. pdbs is a list of class `"3dalign"` containing aligned PDB structure data. This is the output of running `read.fasta.pdb` with the `aln` object. The coordinates are fitted onto the first structure based on `"core"` positions using the funciton `pdbfit`.
2. core is a list of class `"core"` obtained by running the function `core.find` on the `pdbs` object.
3. annotation is a character matrix describing the nucleotide state and bound ligand species for each structure in `pdbs`.

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` exmpales, 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.

`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", het=FALSE, 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`.
`het`	logical, if TRUE ‘HETATM’ records from full PDB files are written to output superposed PDB files. Only required if `full.pdbs` is TRUE.
`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 multicore package 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("1hel")

## 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))
```

`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")
```

`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,
     verbose = TRUE, ncore = 1)
```

Arguments

`ids`	A character vector of one or more 4-letter PDB codes/identifiers of the files to be downloaded.
`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.
`verbose`	print details of the reading process.
`ncore`	number of CPU cores used to do the calculation. `ncore>1` requires multicore package installed.

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`

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)
```

`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.s
```

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 multicore package 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)
```

`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("1hel")

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

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

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'?*

Description

Checks whether its argument is an object of class 'pdb'.

Usage

```
is.pdb(x)
```

Arguments

x	an R object to be tested

Details

Tests if x is an object of class 'pdb', i.e. if x has a "class" attribute equal to `pdb`.

Value

TRUE if x is an object of class 'pdb' and FALSE otherwise

See Also

`read.pdb`

Examples

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

# Print structure summary
atom.select(pdb)

# Select all C-alpha atoms with residues numbers between 65 and 143
ca.inds <- atom.select(pdb, resno=65:143, elety="CA")
is.select(ca.inds)
```

`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.

`load.enmff` *ENM Force Field Loader*

Description

Load force field for elastic network normal mode calculation.

Usage

```
load.enmff(ff = 'calpha')
ff.calpha(r, ...)
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.
`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("1hel")

## 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
a <- mktrj(pc.xray)

detach(transducin)
```

`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("1hel")

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

## Visualize modes
m7 <- mktrj(modes, mode=7, file="mode_7.pdb")
```

`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
a <- mktrj(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(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(pc.xray, pc=3, file="pc3.pdb",
           resno = pdbs$resno[1, gaps.res$f.inds],
           resid = aa123(pdbs$ali[1, gaps.res$f.inds]) )

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)
```

nma — *Normal Mode Analysis*

Description

Perform elastic network model (ENM) C-alpha normal modes calculation of a protein structure.

Usage

```
nma(pdb, inds = NULL, ff = 'calpha', pfc.fun = NULL, mass = TRUE,
    temp = 300.0, keep = NULL, hessian = NULL, ... )

build.hessian(xyz, pfc.fun, aa.mass = NULL, fc.weights = NULL, sse = NULL,
    sequ = 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.
`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.
`xyz`	a numeric vector of Cartesian coordinates.
`aa.mass`	a numeric vector of amino acid residue masses as obtained from function `aa2mass`.
`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`.
`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()` 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("1hel")

## 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", het2atom=TRUE)
## 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

```
nma.pdbs(pdbs, fit=TRUE, full=FALSE, rm.gaps=TRUE,
         outpath="pdbs_nma", ...)

## 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.
`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.
`outpath`	character string specifing the output directory to which the PDB structures should be written.
`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 20 non-trivial eigenvectors ('U') of the 'nma' object).
`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

`nma`, `pdbaln`, `read.fasta.pdb`, `rmsip`

Examples

```
## 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)

## Normal mode analysis on aligned data
all.modes <- nma.pdbs(pdbs, rm.gaps=FALSE)

## Plot fluctuation data
plot.bio3d(all.modes$fluctuations[1,], type="o", xlab="Residue Position")
lines(all.modes$fluctuations[2,], type='o', col=2)
lines(all.modes$fluctuations[3,], type='o', col=3)

## Remove gaps from output
all.modes <- nma.pdbs(pdbs, rm.gaps=TRUE)
```

```
## Compare modes between two structures
rmsip(all.modes$U.subs[,,1], all.modes$U.subs[,,2])
```

`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("1hel")

## 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.project` *Project Data onto Principal Components*

Description

Projects data onto principal components.

Usage

```
pca.project(data, pca, angular = FALSE, fit = FALSE, ...)
pca.z2xyz(z.coord, pca)
pca.xyz2z(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 <- pca.project(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)
```

`pca.tor` *Principal Component Analysis*

Description

Performs principal components analysis (PCA) on torsion angle `data`.

Usage

```
pca.tor(data, subset = rep(TRUE, nrow(as.matrix(data))))
```

Arguments

`data`	numeric matrix of torsion angles 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 `data` matrix. Note: the full `data` is projected onto this subspace.

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

```
pca.xyz(xyz, subset = rep(TRUE, nrow(as.matrix(xyz))))
```

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.

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 normalised 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

`plot.pca`, `mktrj.pca`, `pca.tor`, `pca.project`

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 Read previously saved transducin data
data(transducin)
attach(transducin, warn.conflicts=FALSE)

# Previously fitted coordinates based on sub 1.0A^3 core
xyz <- pdbs$xyz

# Alternatively fit structures onto sub 0.5A^3 core
#xyz <- fit.xyz( fixed = pdbs$xyz[1,],
#                mobile = pdbs,
#                fixed.inds  = core$c0.5A.xyz,
#                mobile.inds = core$c0.5A.xyz)

# 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])

# 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.bio3d(gaps.res$f.ind, pc.xray$au[,1],
#           xlab="Alignment Position", ylab="PC1 (A)")

## Plot loadings in relation to reference structure "1TAG_A"
pdb <- read.pdb("1tag")
sse <- dssp(pdb, resno=FALSE)

ind <- grep("1TAG", pdbs$id)
res.ref <- which(!is.gap(pdbs$ali[ind,]))
res.ind <- which(res.ref %in% 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=sse, ylab="PC1 (A)")
plot.bio3d(res.ind, pc.xray$au[,2], sse=sse, ylab="PC2 (A)")
plot.bio3d(res.ind, pc.xray$au[,3], sse=sse, 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)
```

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.

Details

Given a list of PDB IDs and query terms, this function will download the required information from PDB, remove redundant information and return a matrix 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

Examples

```
# Fetch all annotation terms
pdbanno <- pdb.annotate(c("6Q21_B", "1NVW", "1P2U_A"))

# Access terms, e.g. ligand names:
pdbanno$ligandName

# Fetch only specific terms
pdb.annotate(c("6Q21_B", "1NVW", "1P2U_A"),
             anno.terms = c("ligandId", "experimentalTechnique",
                            "publicationYear", "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 multicore package 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 `"3dalign"` 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 `"3dalign"` 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.

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 `"3dalign"` 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)
```

`pdbs2pdb` *PDBs to PDB Converter*

Description

Convert a list of PDBs from an `"3dalign"` object to a list of `pdb` objects.

Usage

```
pdbs2pdb(pdbs, inds = NULL, rm.gaps = FALSE)
```

Arguments

`pdbs`	a list of class `"3dalign"` 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 `"3dalign"` 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", verbose = FALSE, ...)
```

Arguments

`pdb.files`	a character vector of PDB file names.
`ids`	a character vector of PDB and chain identifiers (‘pdbId_chainId’). Used for filtering chain IDs for output.
`path`	output path for chain-split files.
`verbose`	logical, if TRUE details of the PDB header and chain selections are printed.
`...`	additional arguments to `read.pdb`. Useful e.g. for parsing multi model PDB files, or include HETATM records in the split files.

Details

This function will produce single chain PDB files from multi-chain input files. By default all separate filenames are returned. To avoid this behaviour ‘ids’ can be provided to filter the output (e.g. to fetch only chain C, of a PDB object with additional chains A+B). See examples for details.

Multi model atom records will be splitted into individual PDB files if `multi=TRUE`, else they are omitted.

Value

Returns a character vector of 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

```
  ## Fetch PDBs
  pdb.ids <- c("1YX5", "3NOB", "1P3Q", "2D3G", "1F9J")
  outdir <- paste(tempdir(), "/raw_pdbs", sep="")
  raw.files <- get.pdb(pdb.ids, path = outdir)

  ## Split PDBs by chain ID and multi-model records
  all.files <- pdbsplit(raw.files,
            path =paste(outdir, "/split_chain", sep=""), multi=TRUE )

  list.files(paste(outdir, "/split_chain", sep=""))

## Not run:
  ## desired pdbID-chainID combinations
  ## for the last entry (1f9j), fetch all chains
  ids <- c("1YX5_A", "3NOB_B", "1P3Q_Q", "2D3G_A", "1F9J")
```

```
    ## Same as above, but return only selected pdbID-chainID combinations
    sel.files <- pdbsplit(raw.files, ids,
              path =paste(outdir, "/split_chain", sep=""), multi=TRUE )

## End(Not run)
```

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
                 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=110, mar=c(4, 4, 1, 2), cex.lab=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.
`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.lab`	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.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, xla
```

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=
```

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, entropy = FALSE, col = NULL,
        xlab = "Residue Position", ylab = "Fluctuations",
        mar = c(4, 5, 2, 2), ...)
```

Arguments

`x`	the results of ensemble NMA obtained with `nma.pdbs`.
`pdbs`	an object of class '3dalign' in which the 'enma' object `x` was obtained from. If provided SSE data of the first structure of `pdbs` will drawn.
`entropy`	logical, if TRUE entropy and fluctuation variance is plotted.
`col`	a character vector of plotting colors.
`xlab`	a label for the x axis.
`ylab`	a label for the y axis.
`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.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

```
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=FALSE)

## Plot fluctuations
plot.enma(all.modes, pdbs)
```

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("1hel")

## 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("1hel")

## 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.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)
```

`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 "mypdb-dir/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 `"3dalign"` 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.

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"))

# Sequence names/ids
aln$id

# Alignment positions 335 to 339
aln$ali[,33:39]

# Sequence d1bg2__
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 = "",
               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.
`ncore`	number of CPU cores used to do the calculation. `ncore>1` requires multicore package 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 `"3dalign"` 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.

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", het2atom=T)

## 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, het2atom=FALSE, 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.
`rm.insert`	logical, if TRUE PDB insert records are ignored.
`rm.alt`	logical, if TRUE PDB alternate records are ignored.
`het2atom`	logical, if TRUE HETATM PDB records are stored as ATOM records and returned in the output as such, this should be used with caution.
`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 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.
`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".
`calpha`	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("1bg2")

## Read a PDB file from those included with the package
###pdb <- read.pdb( system.file("examples/1bg2.pdb", package="bio3d") )

## Print data for the first atom
pdb$atom[1,]

## Look at the first het atom
pdb$het[1,]

## 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")])

## Not run:
## Print SSE data (for helix and sheet),
##  see also dssp and stride functions
pdb$helix
pdb$sheet$start

## Print SEQRES data
pdb$seqres

## SEQRES as one leter 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]


## 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")

## 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 = 50000, 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. 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( "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 multicore package 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.dcd(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 multicore package 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 multicore package 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(modes.a, modes.b, subset=10, row.name="a", col.name="b")
```

Arguments

`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`.
`subset`	the number of modes to consider.
`row.name`	prefix name for the rows.
`col.name`	prefix name for the columns.

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.

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.xyz, nma, overlap

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

A data frame containing the following columns:

1. aa1 one-letter code of amino acid residue i
2. aa2 one-letter code of amino acid residue j
3. min minimum distance in which the corresponding spring force constant is value
4. max maximum distance in which the corresponding spring force constant is value
5. k spring force constant valid for the corresponding distance range and amino acid pair.

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)
```

---

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, 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.
exefile	file path to the 'MUSCLE' program on your system (i.e. how is 'MUSCLE' invoked).
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.

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

read.fasta, read.fasta.pdb, seqbind

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)

##-- 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)

## Do further analysis...
## Note that all the above can be done with the pdbaln() function.
## pdbs <- pdbaln( c("first.pdb","second.pdb", "third.pdb") )



##-- 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

```
##- 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)
```

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 and/or matrices 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, 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.
`ncore`	number of CPU cores used to do the calculation. `ncore>1` requires multicore package 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)
```

---

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("1hel")
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

```
     ## Stucture of PKA:
     a <- read.pdb("1cmk")

     ## Stucture of PKB:
     b <- read.pdb("2jdo")

     ## Align and fit b on to a:
     aln <- struct.aln(a, b)

     ## 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)

## 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)

##-- 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")
```

`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")
```

```
## Not run:
##-- 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)

## End(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)
## 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"])
```

`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.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, type="pymol", outprefix="corr",
          launch=FALSE)
```

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.
`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.

Details

This function generates a PyMOL (python) script or a PDB that will draw colored lines between (anti)correlated residues.

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("1hel")

## 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)
```

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.

Details

This function generates a PyMOL (python) script for drawing mode vectors on a PDB structure.

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("1hel")

## 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, e
```

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" )

write.fasta(list( id=c("seqres"),ali=s ), file="eg.fa")
write.fasta(list( id=c("atom"),ali=a ), file="eg.fa", 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]

write.fasta(aln, file="eg2.fa")
```

---

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, resno = NULL, resid =
NULL, eleno = NULL, elety = NULL, chain = NULL, insert = NULL, alt = NULL,
o = NULL, b = NULL, segid = NULL, elesy = NULL, charge = NULL, het = FALSE,
append = FALSE, verbose = FALSE, chainter = FALSE, end = TRUE, print.segid = FAL
```

Arguments

`pdb`	a PDB structure object obtained from `read.pdb`.
`file`	the output file name.
`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.
`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.
`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 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
write.pdb(pdb=pdb, resno = nums, file="eg.pdb")
```

`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
write.pdb(pdb=pdb, resno = nums, file="eg.pdb")
```

Index