The bio3d Package

November 13, 2012

Title Biological Structure Analysis

Version 1.1-5

Author Barry Grant

Suggests ncdf, lattice, grid, bigmemory

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 version 2 or newer

URL http://thegrantlab.org/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:	1.1-5
Date:	2012-11-13
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 (`pdb.summary`), 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`, `identity`, `entropy`, `consensus`) and structural (`pdbaln`, `rmsd`, `rmsf`, `core.find`) data.

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`, `seq.pdb`, `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/.
Or its alternate site:
http://mccammon.ucsd.edu/~bgrant/bio3d/.

Author(s)

Barry Grant <bjgrant@umich.edu>

References

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

Examples

```
## Not run:
help(package="bio3d")  # list the functions within the package
lbio3d()  # list bio3d function names only

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

##-- 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 <- seq.pdb(pdb)

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



## Reorient and write a new PDB file
xyz <- orient.pdb(pdb)
write.pdb(pdb, xyz = xyz, file = "mov1.pdb")


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


##-- Secondary structure assignment with DSSP
sse <- dssp(pdb, resno=FALSE)

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

```
##-- Search for related structures in the PDB database
blast <- blast.pdb( seq.pdb(pdb) )
hits  <- plot.blast(blast)
head(hits$hits)
## Download these with function "get.pdb()"
## Split by chain with function "split.pdb()"
## and then align with function "pdbaln()"
rawpdbs <- get.pdb(hits$pdb.id, "PDB_downloads")
split.pdb(rawpdbs)
hitfiles <- paste("split_chain/", hits$pdb.id, ".pdb", sep="")
hitfiles <- hitfiles[-grep("KIN",hitfiles)]
pdbs <- pdbaln(hitfiles)
xyz  <- fit.xyz( pdbs$xyz[1,], pdbs, full.pdbs=TRUE, het=TRUE)

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

## Entropy score for alignment positions
h <- entropy(aln)    # see also the "conserv()" function

# Alignment consensus
con <- consensus(aln)

## Plot of residue conservation (similarity) with secondary structure
plot.bio3d( conserv(aln)[!is.gap(aln$ali[1,])],  sse=sse )

## 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, pdbext = ".ent",
               pdb.path = system.file("examples",package="bio3d") )


##-- 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
gaps <- gap.inspect(pdbs$xyz)

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


##-- RMSD
```

```
rmsd(pdbs$xyz[1,gaps$f.inds], pdbs$xyz[,gaps$f.inds])
rmsd(pdbs$xyz[1,gaps$f.inds], pdbs$xyz[,gaps$f.inds], fit=TRUE)



##-- Rigid 'core' identification
aln <- read.fasta(system.file("examples/kinesin_xray.fa",package="bio3d"))
pdb.path = system.file("examples/",package="bio3d")
pdbs <- read.fasta.pdb(aln, pdb.path = pdb.path, pdbext = ".ent")

core <- core.find(pdbs,
                  #write.pdbs=TRUE,
                  rm.island=TRUE,
                  verbose=TRUE)

## Plot core volume vs length
plot(core)

## Core fit the structures (superpose on rigid zones)
xyz <- fit.xyz( fixed = pdbs$xyz[1,],
                mobile = pdbs,
               #  full.pdbs = TRUE,
               #  pdb.path = pdb.path,
               #  pdbext = ".ent",
               #  outpath = "fitlsq/",
                 fixed.inds  = core$c0.5A.xyz,
                 mobile.inds = core$c0.5A.xyz)


##-- PCA of structures
cut.seqs <- which(pdbs$id %in% c("d1n6mb_","d1ry6a_"))
# Ignore gap containing positions
gaps.res <- gap.inspect(pdbs$ali[-cut.seqs,])
gaps.pos <- gap.inspect(pdbs$xyz[-cut.seqs,])


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

## Plot results
plot(pc.xray)

#x11()
plot.pca.loadings(pc.xray$U)

## 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]) )




##-- 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.loadings(pc.trj)

## 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: sdm, 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)
```

`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://mccammon.ucsd.edu/~bgrant/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/kinesin_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 "eg.html" 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.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)
pdb.summary(pdb)
```

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/`.
`chain`	a character vector of chain identifiers.
`resno`	a numeric or character vector of residue numbers.
`resid`	a character vector of chain 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. `pdb.summary` is a simple alias to `atom.select` with `string=NULL` and `verbose=T`.

Value

Returns a list of class `"selection"` 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)
```

atom2xyz *Convert Between Atom and xyz Indices*

Description

Basic function to return the Calpha xyz indices given their corresponding Calpha only atom indices.

Usage

```
atom2xyz(num)
```

Arguments

num	a numeric vector of atom indices.

Value

A numeric vector of xyz 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))

## For xyz2atom:
unique( ceiling(xyz.ind/3) )
```

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( seq.pdb(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

`plot.dmat`, `cm.colors`, `colors`, `palette`, `hsv`, `rgb`, `gray`, `col2rgb`

Examples

```
# Color a distance matrix
pdb <- read.pdb( system.file("examples/1bg2.pdb", package = "bio3d") )
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 a Given Protein Structure.

Usage

```
cmap(xyz, grpby=NULL, dcut = 4, scut = 3, mask.lower = TRUE)
```

Arguments

xyz	numeric vector of xyz 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).
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.
mask.lower	logical, if TRUE the lower matrix elements (i.e. those below the diagonal) are returned as NA.

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 and Traj files
pdb <- read.pdb( system.file("examples/hivp.pdb", package="bio3d") )
##trj <- read.dcd( system.file("examples/hivp.dcd", 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:
## 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)
```

---

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 kinesin alignment
aln <- read.fasta(system.file("examples/kinesin_xray.fa",package="bio3d"))

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

# Generate consensus for sub-sequence
con <- consensus(aln$ali[,330:474])
print(con$seq)


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

# 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/kinesin_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 = TRUE, rm.wat =
```

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 (version2.2) see:
http://www.rcsb.org/pdb/file_formats/pdb/pdbguide2.2/guide2.2_frame.html.

See Also

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

Examples

```
# Read a PDB file
pdb <- read.pdb( system.file("examples/1bg2.pdb", package="bio3d") )
pdb.summary(pdb)
# Convert to CHARMM format
new <- convert.pdb(pdb, type="charmm")
pdb.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/")
```

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.

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:
##--  Read kinesin alignment and respective PDB structures
aln <- read.fasta(system.file("examples/kinesin_xray.fa",package="bio3d"))
pdb.path = system.file("examples",package="bio3d")
pdbs <- read.fasta.pdb(aln, pdb.path = pdb.path, pdbext = ".ent")

## End(Not run)

##-- Or read previously saved kinesin data
```

```
data(kinesin)
attach(kinesin)

## Raw RMSD before superposition
gaps <- gap.inspect(pdbs$xyz)
rmsd( pdbs$xyz[,gaps$f.inds] )

## RMSD after superposition on all positions
#rmsd(pdbs$xyz[,gaps$f.inds],fit=TRUE)

## Run core.find
core <- core.find(pdbs,
                  #write.pdbs = TRUE,
                  verbose=TRUE)


## Plot volume vs length
plot(core)

## Print 0.5A^3 core and store indices
inds <- print(core, vol=0.5)

## Fit structures onto first structure based on core indices (inds$xyz)
xyz <- fit.xyz( fixed = pdbs$xyz[1,],
                mobile = pdbs,
                fixed.inds  = inds$xyz,
                mobile.inds = inds$xyz)


# RMSD after superposition on 'core' positions
rmsd( xyz[,gaps$f.inds] )


## Not run:
# Fit structures and write out 'full' structures
xyz <- fit.xyz( fixed = pdbs$xyz[1,],
                mobile = pdbs,
                fixed.inds  = core$c0.5A.xyz,
                mobile.inds = core$c0.5A.xyz,
                pdb.path = system.file("examples/",package="bio3d"),
                pdbext = ".ent",
                outpath = "fitlsq/",
                full.pdbs = TRUE)

gaps  <- unique(which( is.na(xyz),arr.ind=TRUE )[,2])

# core fitted RMSD
rmsd(xyz[1,-gaps], xyz[,-gaps])

# original RMSD
rmsd(xyz[1,-gaps], xyz[,-gaps], fit=TRUE)


##-- Try 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(xyz, reference = apply(xyz, 2, mean))
```

Arguments

`xyz`	a numeric matrix of Cartesian coordinates with a row per structure/frame.
`reference`	The reference structure about which displacements are analysed.

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)
```

`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)
```

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.

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 (normalized) of the structural difference.

Author(s)

Lars Skjaerven

References

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

See Also

`overlap`

Examples

```
data(kinesin)
attach(kinesin)

# 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)

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

dv <- difference.vector( 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)
```

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.

Details

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

Value

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

Note

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

Author(s)

Barry Grant

References

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

See Also

dm, dist

Examples

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

---

dm *Distance Matrix Analysis*

Description

Construct a distance matrix for a given protein structure.

Usage

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

Arguments

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

Details

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

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

Value

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

Note

The input selection can be any character string or pattern interpretable by the function atom.select. For example, shortcuts "calpha", "back", "all" and selection strings of the form /segment/chain/residue number/r 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( system.file("examples/d1bg2__.ent", package = "bio3d") )
k <- dm(pdb,selection="calpha")
filled.contour(k, nlevels = 4)


##--- DDM: Difference Distance Matrix
# Read aligned PDBs
aln <- read.fasta(system.file("examples/kif1a.fa",package="bio3d"))
pdb.path=paste(system.file(package="bio3d"),"/examples/",sep="")
m <- read.fasta.pdb(aln, pdb.path = pdb.path, pdbext = ".ent")

# Get distance matrix
a <- dm(m$xyz[2,])
b <- dm(m$xyz[3,])

# Calculate DDM
c <- a - b

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

plot(c, axis.tick.space=10,
     resnum.1=m$resno[1,],
```

```
     resnum.2=m$resno[2,],
     grid.col="gray",
     xlab="Residue No. (1i6i)", ylab="Residue No. (1i5s)")

## 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)


##--- Extract all-atom contacts
pdb <- read.pdb( system.file("examples/d1bg2__.ent", package = "bio3d") )
l <- dm(pdb,selection="all")
l[upper.tri(l)]=NA  # make top diagonal NA

# Find residues with contacting atoms (<=5 Angstrom)
inds.stru <- which(l<=5, arr.ind=TRUE)

# Find non-consecutive residues (>5 residues sequence separation)
seq.sep  <- abs(as.numeric(pdb$atom[inds.stru[,1],"resno"]) -
               as.numeric(pdb$atom[inds.stru[,2],"resno"]))
inds.seq <- which(seq.sep>5) # seperated by > 5 residues

# All-atom contacts (indices)
inds <- inds.stru[inds.seq,]

# All-atom contacts (now in terms of residue numbers)
tmp <- unique( paste(pdb$atom[inds[,1],"resno"],
                     pdb$atom[inds[,2],"resno"], sep="#") )

contacts <- matrix(as.numeric(unlist(strsplit(tmp,split="#"))),
                   ncol=2, byrow=TRUE )

# Plot residue contacts
freq  <- table(contacts)
xaxis <-as.vector(bounds(as.numeric(names(freq)))[,c(1,2)])
x11()
plot(freq, typ="h", xlab="Residue Number",
     xaxt="n",ylab="Number of Contacts" )
axis(1,at=xaxis,labels=xaxis)

## End(Not run)
```

---

dssp *Secondary Structure Analysis with DSSP*

Description

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

Usage

```
dssp(pdb, exepath = "", resno=TRUE)
```

Arguments

pdb	a structure object of class "pdb", obtained from read.pdb.
exepath	path to the 'DSSP' program on your system (i.e. the directory where 'DSSP' is stored).
resno	logical, if TRUE output is in terms of residue numbers rather than residue index (position in sequence).

Details

This function calls the 'DSSP' 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.

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, stride, torsion.pdb, torsion.xyz, plot.bio3d

Examples

```
## Not run:
# Read a PDB file
pdb <- read.pdb(system.file("examples/d1bg2__.ent", package="bio3d"))
sse <- dssp(pdb)

# 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)
```

`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)
```

`fit.xyz` *Coordinate Superposition*

Description

Coordinate superposition with the Kabsch algorithm.

Usage

```
fit.xyz(fixed, mobile,
        fixed.inds  = NULL,
        mobile.inds = NULL,
        verbose=FALSE,
        pdb.path = "", pdbext = "",
        outpath = "fitlsq/", het=FALSE, full.pdbs=FALSE, ...)

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.
pdb.path	path to the "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.

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.

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

```
## Not run:
##--- Read an alignment & Fit aligned structures
aln  <- read.fasta(system.file("examples/kif1a.fa",package="bio3d"))
pdb.path = system.file("examples",package="bio3d")
pdbs <- read.fasta.pdb(aln, pdb.path = pdb.path, pdbext = ".ent")

## End(Not run)

#-- OR Read previously saved kinesin data & Fit aligned structures
data(kinesin)
attach(kinesin)

gaps <- gap.inspect(pdbs$xyz)

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

# Use indices from 'core.find()' for improved fitting
#core <- core.find(pdbs) # see ?core.find

# 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,
               pdb.path = system.file("examples/",package="bio3d"),
               pdbext = ".ent",
               outpath = "rough_fit/",
               full.pdbs = TRUE)

##--- Fit two PDBs
A <- read.pdb(system.file("examples/d1bg2__.ent",package="bio3d"))
A.ind <- atom.select(A, "///256:269///CA/")

B <- read.pdb(system.file("examples/d2kin.1.ent",package="bio3d"))
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)
```

---

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)
```

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.

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 (version2.2) see:
http://www.rcsb.org/pdb/file_formats/pdb/pdbguide2.2/guide2.2_frame.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) )
pdb.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.ebi.ac.uk/help/formats_frame.html. 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

```
##pdb <- read.pdb( get.pdb("5p21", URLonly=TRUE) )
##blast <- blast.pdb( seq.pdb(pdb), database = "swiss" )
##ids <- plot.blast( blast )
##get.seq(ids)
```

---

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)
```

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 identity.
cutoff	a numeric identity cutoff value ranging between 0 and 1.
verbose	logical, if TRUE print details of the clustering process.

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, identity, entropy, consensus

Examples

```
data(kinesin)
attach(kinesin)
ide.mat <- identity(aln)

# Histogram of pairwise identity values
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 <- identity(aln$ali[k$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
```

---

identity *Percent Identity*

Description

Determine the percent identity scores for aligned sequences.

Usage

```
identity(alignment, normalize=TRUE)
```

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.

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/kinesin_xray.fa",
#                     package = "bio3d") )

## End(Not run)

data(kinesin)
attach(kinesin)

ide.mat <- identity(aln)

# 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
identity( rbind(aln$ali[1,], aln$ali[20,]) )
```

`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/kinesin_xray.fa",
                   package = "bio3d") )

##- Mask gaps with an "X"
xaln <- aln
```

```
xaln$ali[ is.gap(xaln$ali) ]="X"

##- Read a PDB and align its sequence to the existing alignment
pdb <- read.pdb( system.file("examples/1bg2.pdb", package="bio3d") )
pdbseq <- aa321(pdb$atom[pdb$calpha,"resid"])

seqaln.pair(seqbind(xaln$ali[1,], pdbseq))

## End(Not run)
```

`kinesin` *Bio3d Example Data*

Description

These data sets contain the results of running various bio3d functions on example 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)
```

Format

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

1. alnis a list with two components: `ali`, a character alignment matrix, and `id`, a character vector of ids/names. This is the output of running `read.fasta` on the example alignment "kinesin_xray.fa".
2. pdbsis a list of class `"3dalign"` containing aligned PDB structure data. This is the output of running `read.fasta.pdb` with the `aln` object.
3. coreis a list of class `"core"` obtained by running the function `core.find` on the `pdbs` object.
4. xyzis a numeric matrix of "core fitted coordinates" obtained by running the function `fit.xyz` on the `pdbs` object, using the "core atom indices" contained in `core`.
5. pc.xrayis a list object containing PCA results obtained by running the function `pca.xyz` on the "core fitted coordinates" contained in the `xyz` object.
6. ssesecondary structure annotation of a representative kineisn structure, 1bg2, as obtained from the function `dssp`

Source

FASTA alignment and PDB structure data from the bio3d "example" directory. A related but more extensive dataset formed the basis of the work described in (Grant, 2007).

References

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

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.

mktrj.pca *PCA Atomic Displacement Trajectory*

Description

Make a trajectory of atomic displacments along a given principal component.

Usage

```
mktrj.pca(pca = NULL, pc = 1, mag = 1, step = 0.125, file = NULL,  ...)
```

Arguments

pca	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

The molecular graphics software VMD is useful for viewing trajectories see: 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

Examples

```
data(kinesin) # see pca.xyz for details
attach(kinesin)

# Ignore gap containing positions
cut.seqs <- which(pdbs$id %in% c("d1n6mb_","d1ry6a_"))
gaps <- gap.inspect(pdbs$ali[-cut.seqs,])

# Write PC trajectory
a <- mktrj.pca(pc.xray, pc=1, file="pc1.pdb",
               resno = pdbs$resno[1, gaps$f.inds],
               resid = aa123(pdbs$ali[1, gaps$f.inds]) )

b <- mktrj.pca(pc.xray, pc=2, file="pc2.pdb",
               resno = pdbs$resno[1, gaps$f.inds],
               resid = aa123(pdbs$ali[1, gaps$f.inds]) )

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

---

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, seq.pdb

Examples

```
aa.seq <- seq.pdb( read.pdb( get.pdb("4q21", URLonly=TRUE) ) )
motif = c("G....GKS")
motif.find(motif, aa.seq)
```

---

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( system.file( "examples/1bg2.pdb", package = "bio3d") )
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(pca, dv, num.modes=20)
```

Arguments

`pca`	an object of class `"pca"` as obtained from function `pca.xyz`, or alternatively a 3NxM matrix of eigenvectors.
`dv`	a displacement vector of length 3N.
`num.modes`	the number of modes to consider.

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 `pca$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 `pca`.
`overlap.cum`	a numeric vector of the cumulative squared overlap values.

Author(s)

Lars Skjaerven

References

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

See Also

`rmsip`, `pca.xyz`

Examples

```
data(kinesin)
attach(kinesin)

# 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)

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

dv <- difference.vector( 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)
```

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

identity

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)
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.
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

Examples

```
## Not run:
data(kinesin)
gaps.pos <- gap.inspect(pdbs$xyz)

#-- Do PCA without structures 1 and 2
pc.xray <- pca.xyz(xyz[-c(1:2), gaps.pos$f.inds])
d <- pca.project(xyz[1:2, gaps.pos$f.inds], pc.xray)

plot(pc.xray$z[,1], pc.xray$z[,2],col="gray")
points(d[,1],d[,2], col="red")

## 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)

gaps <- gap.inspect(pdbs)
tgap.xyz <- atom2xyz(gaps$t.inds)
fgap.xyz <- atom2xyz(gaps$f.inds)
tor <- t(apply( pdbs$xyz[,fgap.xyz], 1, torsion.xyz, atm.inc=1))
pc.tor <- pca.tor(tor[,-c(1,219,220)])
#plot(pc.tor)
plot.pca.loadings(pc.tor)

## 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 kinesin alignment and structures
aln <- read.fasta(system.file("examples/kinesin_xray.fa",package="bio3d"))
pdb.path = system.file("examples/",package="bio3d")
pdbs <- read.fasta.pdb(aln, pdb.path = pdb.path, pdbext = ".ent")

# Find core
```

```
core <- core.find(pdbs,
                  #write.pdbs = TRUE,
                  verbose=TRUE)

# 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)

## End(Not run)

#-- OR Read previously saved kinesin data
data(kinesin)
attach(kinesin)

# Remove outlier structures from kinesin alignment
cut.seqs <- which(pdbs$id %in% c("d1n6mb_","d1ry6a_"))

# Ignore gap containing positions
gaps.res <- gap.inspect(pdbs$ali[-cut.seqs,])
gaps.pos <- gap.inspect(pdbs$xyz[-cut.seqs,])


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

# Plot results (conformer plots & scree plot)
plot(pc.xray)

## 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 "d1bg2__"
res.ref <- which(!is.gap(pdbs$ali["d1bg2__",]))
res.ind <- which(res.ref %in% gaps.res$f.ind)
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)")



## 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)
```

---

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, pqr = FALSE, ...)
```

Arguments

files	a character vector of PDB file names.
pqr	logical, if TRUE the input structures are assumed to be in PQR format.
...	extra arguments passed to seqaln function.

Details

This wrapper function calls the underlying functions read.pdb, seq.pdb, 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, seq.pdb, seqaln, read.fasta,read.fasta.pdb, core.find, fit.xyz, read.all

Examples

```
## Not run:
files <- get.pdb(c("4q21","5p21"), URLonly=TRUE)
pdbaln(files)

## 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

```
plot.bio3d(x, y = NULL, type = "h", main = "", sub = "", xlim = NULL, ylim = NULL,
                ylim2zero = TRUE, xlab = NULL, ylab = NULL, axes = TRUE,
                ann = par("ann"), col = par("col"), sse = NULL, top = TRUE, bot = TRUE,
                helix.col = "gray20", sheet.col = "gray80", sse.border = FALSE, ...)
```

Arguments

x	the x coordinates for the plot. Any reasonable way of defining the coordinates is acceptable. See the function 'xy.coords' for details.
y	the y coordinates for the plot, see above.
type	one-character string giving the type of plot desired. The following values are possible, (for details, see 'plot'): 'p' for points, 'l' for lines, 'o' for overplotted points and lines, 'b', 'c') for points joined by lines, 's' and 'S' for stair steps and 'h' for histogram-like vertical lines. Finally, 'n' does not produce any points or lines.
main	a main title for the plot, see also 'title'.
sub	a sub-title for the plot.
xlim	the x limits (x1,x2) of the plot. Note that x1 > x2 is allowed and leads to a reversed axis.
ylim	the y limits of the plot.
ylim2zero	logical, if TRUE the y-limits are forced to start at zero.
xlab	a label for the x axis, defaults to a description of 'x'.
ylab	a label for the y axis, defaults to a description of 'y'.
axes	a logical value indicating whether both axes should be drawn on the plot. Use graphical parameter 'xaxt' or 'yaxt' to suppress just one of the axes.

ann	a logical value indicating whether the default annotation (title and x and y axis labels) should appear on the plot.
col	The colors for lines and points. Multiple colours can be specified so that each point is given its own color. If there are fewer colors than points they are recycled in the standard fashion. Lines are plotted in the first colour specified.
sse	secondary structure object as returned from dssp or stride.
top	logical, if TRUE rectangles for each sse are drawn towards the top of the plotting region.
bot	logical, if TRUE rectangles for each sse are drawn towards the bottom of the plotting region.
helix.col	The colors for rectangles representing alpha helices.
sheet.col	The colors for rectangles representing beta strands.
sse.border	The border color for all sse rectangles.
...	other graphical parameters.

Details

See the functions 'plot.default', dssp and stride for further details.

Value

Called for its effect.

Note

Be sure to check the correspondence of your 'sse' object with the 'x' values being plotted as no internal checks are performed.

Author(s)

Barry Grant

References

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

See Also

plot.default, dssp, stride

Examples

```
##-- SSE setup --#
pdb <- read.pdb(  system.file("examples/1bg2.pdb", package="bio3d") )
sse <- dssp(pdb, resno=FALSE)

##-- Or load previous dataset
data(kinesin)
attach(kinesin)

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

```
## Not run:
## Tidy short helices
torm <- sse$helix$length < 4
sse$helix$length <- sse$helix$length[!torm]
sse$helix$start <- sse$helix$start[!torm]
sse$helix$end <- sse$helix$end[!torm]


##-- Read a FASTA alignment file
aln <- read.fasta( system.file("examples/kinesin_xray.fa",package="bio3d") )

## Plot of residue conservation (similarity) with secondary structure
plot.bio3d( conserv(aln)[!is.gap(aln$ali[1,])],  sse=sse )



##- Gaps relative to rep.stru
data(kinesin)
rep.stru <- 1
gap.atom.inds <- which(aln$ali[rep.stru,] == "-")
gap.xyz.inds  <- atom2xyz(gap.atom.inds)

## RMSF and B-factors
coords <- xyz[,-gap.xyz.inds]
bfac <- pdbs$b[,-gap.atom.inds]
r <- rmsf(coords)

##- Plot
plot.bio3d(r)
plot.bio3d(r, type="o", sse=sse)
plot.bio3d(r, sse=sse, helix.col="purple", sheet.col="orange")


## 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

```
plot.blast(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( seq.pdb(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( seq.pdb(read.pdb( get.pdb("2BN3", URLonly=TRUE) )))
raw.hits <- plot(blast)
top.hits <- plot(blast, cut.seed=20)

head(top.hits$pdb.id)
pdbFiles <- get.pdb(top.hits$pdb.id, dir="downloadedPDBs/")
split.pdb(pdbFiles, dir="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

```
plot.core(x, y = NULL, type = "h", main = "", sub = "", xlim = NULL, ylim = NULL, xlab = "Core S
```

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:
##-- Read kinesin alignment and structures
aln <- read.fasta(system.file("examples/kinesin_xray.fa",package="bio3d"))
pdb.path = system.file("examples/",package="bio3d")
pdbs <- read.fasta.pdb(aln, pdb.path = pdb.path, pdbext = ".ent")

## Find core
core <- core.find(pdbs,
                  #write.pdbs = TRUE,
                  verbose=TRUE)

## End(Not run)

## Not run:
##-- OR read previously saved kinesin data
data(kinesin)
attach(kinesin)

col=rep("black", length(core$volume))
col[core$volume<2]="pink"; col[core$volume<1]="red"
plot(core, col=col)

print(core)

## End(Not run)
```

---

plot.dccm *DCCM Plot*

Description

Plot a dynamical cross-correlation matrix.

Usage

```
plot.dccm(x, sse = NULL, colorkey = TRUE, at = c(-1, -0.75, -0.5, -0.25, 0.25, 0.5, 0.75, 1), ma
```

Arguments

x	a numeric matrix of atom-wise cross-correlations as output by the 'dccm' function.
sse	secondary structure object as returned from dssp or stride.
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.
...	additional graphical parameters for image.

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 'cij' 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.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)

cij <- dccm(xyz)
plot.dccm(cij)

## SSE axis labs
sse <- dssp(pdb, resno=FALSE)
gridpoints <- c( sse$helix$start[sse$helix$length>4],
                 sse$sheet$start[sse$sheet$length>4])

scales <- list(at=gridpoints,
               labels=gridpoints, alternating=1, ticks=F)

plot.dccm(cij, sse=sse, scales=scales)
##add.grid(sse)

## Plot with changed colors
plot.dccm(cij, col.regions=rev(heat.colors(12)))


## End(Not run)
```

`plot.dmat` *Plot Distance Matrix*

Description

Plot a distance matrix (DM) or a difference distance matrix (DDM).

Usage

```
plot.dmat(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(system.file("examples/1bg2.pdb",package="bio3d"))

# 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")


# Read alignment
aln <- read.fasta(system.file("examples/kif1a.fa",package="bio3d"))
pdb.path = system.file("examples/",package="bio3d")
pdbs <- read.fasta.pdb(aln, pdb.path = pdb.path, pdbext = ".ent")

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

# DDM: difference distance matrix (C-alpha)
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="gray",
     xlab="Residue No. (1bg2)", ylab="Residue No. (2kin)")

## End(Not run)
```

`plot.pca` *Plot PCA Results*

Description

Produces a z-score plot (conformer plot) and an eigen spectrum plot (scree plot).

Usage

```
plot.pca(x, pch = 16, col = par("col"), cex=0.8, mar=c(4, 4, 1, 1),...)
plot.pca.scree(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, ...)
plot.pca.score(x, start.row = 1, end.row = nrow(x), 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.
start.row	row index of the first conformer to colour.
end.row	row index of the last conformer to colour.
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(kinesin)
attach(kinesin)

plot(pc.xray)

## change colors
plot(pc.xray,col=rainbow(length(pc.xray$z[,1])), cex=1.5)

## add labels

## color by seq identity
```

---

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

```
plot.pca.loadings(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(kinesin)
attach(kinesin)
plot.pca.loadings(pc.xray$U)
```

print.core *Printing Core Positions and Returning Indices*

Description

Print method for core.find objects.

Usage

```
print.core(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:
##-- Read kinesin alignment and structures
aln <- read.fasta(system.file("examples/kinesin_xray.fa",package="bio3d"))
pdb.path = system.file("examples/",package="bio3d")
pdbs <- read.fasta.pdb(aln, pdb.path = pdb.path, pdbext = ".ent")

## Find core
core <- core.find(pdbs,
```

```
                  #write.pdbs = TRUE,
                  verbose=TRUE)

## End(Not run)

##-- OR read previously saved kinesin data
data(kinesin)
attach(kinesin)

col=rep("black", length(core$volume))
col[core$volume<2]="pink"; col[core$volume<1]="red"
plot(core, col=col)

inds.1 <- print(core, vol=1)
inds.0.5 <- print(core, vol=0.5)
```

`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, pdb.path = "", pdbext = "", sel = NULL, ...)
```

Arguments

`aln`	an alignment data structure obtained with `read.fasta`.
`pdb.path`	path to PDB files.
`pdbext`	the file name extention of the PDB files.
`sel`	a selection string detailing the atom type data to store (see function store.atom)
`...`	other parameters for `read.pdb`.

Details

The input `aln`, produced with `read.fasta`, must have identifers (i.e. sequence names) that match the PDB file names. For example the sequence corresponding to the structure file "mypdbdir/1bg2.pdb" should have the identifer 'mypdbdir/1bg2.pdb' or '1bg2' if input 'pdb.path' 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.
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
## Not run:
# Read sequence alignment
file <- system.file("examples/kif1a.fa",package="bio3d")
aln  <- read.fasta(file)

# Read aligned PDBs
pdb.path = system.file("examples",package="bio3d")
sel <- c("N", "CA", "C", "O", "CB", "*G", "*D",  "*E", "*Z")
pdbs <- read.all(aln, pdb.path = pdb.path, pdbext = ".ent", sel=sel)

atm <- colnames(npdbs$all)
ca.ind  <- which(atm == "CA")
core <- core.find(pdbs)
core.ind <- c( matrix(ca.ind, nrow=3)[,core$c0.5A.atom] )

nxyz <- fit.xyz(npdbs$all[1,], npdbs$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=npdbs$all.elety[1,xyz2atom(ngap.col$f.inds)],
                resid=npdbs$all.resid[1,xyz2atom(ngap.col$f.inds)],
                resno=npdbs$all.resno[1,xyz2atom(ngap.col$f.inds)] )



## 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.charmm.org/document/Charmm/c32b2/io.html.

See Also

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

Examples

```
##
## crd <- read.crd("somefile.crd")
## crd$atom[crd$calpha,c("x","y","z")]
## write.pdb(crd, file="somefile.pdb")
##
```

---

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)
```

Arguments

trjfile	name of trajectory file to read.
big	logical, if TRUE attempt to read large files into a big.matrix object
verbose	logical, if TRUE print details of the reading process.

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.

Value

A numeric matrix of xyz coordinates with a frame/structure per row and a Cartesian coordinate 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 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
fit.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=fit.xyz)

## Not run:
# Pairwise RMSD of trj frames for positions 47 to 54
flap.inds <- atom.select(pdb,"///47:54///CA/")
p <- rmsd(fit.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.ebi.ac.uk/help/formats_frame.html`. 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, pdb.path = "", pdbext = "", ...)
```

Arguments

`aln`	an alignment data structure obtained with `read.fasta`.
`pdb.path`	path to PDB files.
`pdbext`	the file name extention of the PDB files.
`...`	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 `"3daligh"` 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

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
pdb.path = system.file("examples",package="bio3d")
pdbs <- read.fasta.pdb(aln, pdb.path = pdb.path, pdbext = ".ent")

# Structure/sequence names/ids
pdbs$id

# Alignment positions 335 to 339
pdbs$ali[,335:339]
pdbs$resno[,335:339]
pdbs$b[,335:339]

# Alignment C-alpha coordinates for first structure
pdbs$xyz[1,]
```

`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)
```

Arguments

`trjfile`	name of trajectory file to read.
`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.

Details

Reads a AMBER netCDF format trajectory file with the help of David W. Pierce's (UCSD) ncdf package available from CRAN.

Value

Either a list of trajectory header data or a numeric matrix of xyz coordinates with a frame/structure per row and a Cartesian coordinate per column.

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://amber.scripps.edu/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 = 50000, multi = FALSE, rm.insert = FALSE,
         rm.alt = TRUE, het2atom=FALSE, verbose = TRUE)
print.pdb(x, ...)
```

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. 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.
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.
...	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.
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 (version2.2) see:
http://www.rcsb.org/pdb/file_formats/pdb/pdbguide2.2/guide2.2_frame.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", seq.pdb(pdb))
seq.pdb(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 (version2.2) see:
http://www.rcsb.org/pdb/file_formats/pdb/pdbguide2.2/guide2.2_frame.html.

See Also

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

Examples

```
# Read a PDB file
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
pdb$atom[, 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 (version2.2) see:
http://www.rcsb.org/pdb/file_formats/pdb/pdbguide2.2/guide2.2_frame.html.

See Also

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

Examples

```
# Read a PDB file
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
pdb$atom[, 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")
```

`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)
```

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.

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"))
pdb.path = system.file("examples",package="bio3d")
pdbs <- read.fasta.pdb(aln, pdb.path = pdb.path, pdbext = ".ent")

# 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)
```

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.

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)
k <- rmsd.filter(pdbs,cutoff=0.5)
aln$id[k$ind]
plot(k$tree, ylab="RMSD")
abline(h=0.5, col="gray")

## 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(kinesin)
attach(kinesin)

r <- rmsf(xyz)
plot( r[aln$ali["d1bg2__",]!="-"], typ="h", ylab="RMSF (A)")
```

`rmsip` *Root Mean Square Inner Product*

Description

Calculate the RMSIP between two mode subspaces.

Usage

```
rmsip(pca.a, pca.b, subset=10, row.name=NULL, col.name=NULL)
```

Arguments

`pca.a`	an object of class `"pca"` as obtained from functions `pca.xyz`.
`pca.b`	an object of class `"pca"` as obtained from functions `pca.xyz`.
`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.

Structure `pca.a$U` and `pca.b$U` should have the same number of columns.

Value

Returns a list 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<e2><80><93>-424.

See Also

`overlap`, `pca.xyz`

Examples

```
data(kinesin)
attach(kinesin)

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

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

## Not run:
# Read in MD trajectory
sim <- NULL
sim$trj <- read.ncdf("300ns_CA.nc")
sim$xyz <- fit.xyz(fixed = pdbs$xyz[1, ], mobile =  sim$trj,
                fixed.inds = core$c1A.xyz, mobile.inds = core$c1A.xyz,
                  full.pdbs = FALSE, het2atom = TRUE)

# Do PCA on the simulation
sim$pc <- pca.xyz(sim$xyz)

# Overlap analysis
o <- overlap(sim$pc, dv, num.modes=50)

# Calculate the RMSIP between the Xray-PCs and the MD-PCs
r <- rmsip(pc.xray, sim$pc, subset=10, row.name="Xray", col.name="MD")

# Plot pairwise overlap values
image(1:10, 1:10, r$overlap, col=gray(50:0/50), zlim=c(0,1),
      xlab="MD-PCs", ylab="X-ray PCs")

# Convergence of eigenvectors
all.rmsip <- c()
intervals <- seq(100,1000, by=100)
for ( i in 1:length(intervals) ) {
   traj.inds <- 1:intervals[i]
   tmp.pc <- pca.xyz(sim$xyz[traj.inds,])
   r <- rmsip(sim$pc, tmp.pc, subset=10)
   all.rmsip <- c(all.rmsip, r$rmsip)
}


## End(Not run)
```

---

seq.pdb *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

```
seq.pdb(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

```
##pdb <- read.pdb( get.pdb("5p21", URLonly=TRUE) )
##seq.pdb(pdb)
```

---

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", exepath = "", 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.
exepath	path to the 'MUSCLE' program on your system (i.e. the directory where 'MUSCLE' is stored).
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.
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

```
## Not run:
aa.1 <- seq.pdb( read.pdb("1bg2") )
aa.2 <- seq.pdb( read.pdb("3dc4") )
aa.3 <- seq.pdb( 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, exepath="", file="aln.fa", protein=TRUE,
       seqgroup=FALSE, refine=FALSE, extra.args="")
```

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.
`exepath`	path to the 'MUSCLE' program on your system (i.e. the directory where 'MUSCLE' is stored).
`file`	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.

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)


##-- Extract sequences
raw <- NULL
for(i in 1:length(files)) {
  pdb <- read.pdb(files[i])
  raw <- seqbind(raw, aa321(pdb$atom[pdb$calpha,"resid"]))
}


##-- Align sequences
```

```
aln <- seqaln(raw, id=basename(files),
              file="seqaln.fa")

##-- Read Aligned PDBs
pdbs <- read.fasta.pdb(aln, pdb.path=pdb.path, pdbext = "")

## 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, file="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 a pair of vector or matrix arguments and combine them row-wise without recycling the shorter argument (as is the case with rbind).

Usage

```
seqbind(a, b, blank = "-")
```

Arguments

a	a vector or matrix.
b	a vector or matrix to be appended to a.
blank	a character to add to the shorter argument, to achieve the same length as the longer argument.

Value

A matrix combining the a and b 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( system.file("examples/1bg2.pdb", package="bio3d") )
b.pdb <- read.pdb( system.file("examples/d1goja_.ent", package="bio3d") )

seqs <- seqbind(aa321(a.pdb$atom[a.pdb$calpha,"resid"]),
                aa321(b.pdb$atom[b.pdb$calpha,"resid"]))

# seqaln(seqs)

## End(Not run)
```

---

split.pdb *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

```
split.pdb(pdb.files, path = "split_chain/")
```

Arguments

pdb.files	a character vector of PDB file names.
path	output path for chain-split files.

Details

This function will produce single chain PDB files from multi-chain input files.

Value

Called for its effect.

Note

To Be Improved.

Author(s)

Barry Grant

References

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

For a description of PDB format (version2.2) see:
http://www.rcsb.org/pdb/file_formats/pdb/pdbguide2.2/guide2.2_frame.html .

See Also

read.pdb, atom.select, write.pdb.

Examples

```
## Not run:
pdbfiles <- list.files()
split.pdb(pdbfiles)
list.files("split_chain")

## 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:
sel <- c("N", "CA", "C", "O", "CB", "*G", "*D",  "*E", "*Z")
pdbs <- read.all(aln, pdb.path = pdb.path, pdbext = ".ent", sel=sel)
core <- core.find(pdbs, rm.island=TRUE)

##-- Core indices
atm <- rep( rep(sel,each=3), ncol(aln$ali))
ca.ind  <- which(atm == "CA")
core.ind <- c( matrix(ca.ind, nrow=3)[,core$c0.5A.atom] )

##-- Core Fit
xyz <- fit.xyz(fixed=tmp[1,],
             mobile = tmp,
             fixed.inds  = core.ind,
             mobile.inds = core.ind)


##-- Examine all-atom data
colnames(tmp) = atm

gap.ind <- unique(which(is.na(tmp),arr.ind=T)[,2])
nogap.ind    <- c(1:ncol(tmp))[-gap.ind]
#ca.nogap.ind <- intersect(ca.ind, nogap.ind)


##-- PCA
xray.pca <- pca.xyz( xyz[,nogap.ind] )

plot(xray.pca)

## End(Not run)
```

`stride` *Secondary Structure Analysis with STRIDE*

Description

Secondary structure assignment according to the method of Frishman and Argos.

Usage

```
stride(pdb, exepath = "")
```

Arguments

`pdb`	a structure object of class '"pdb"', obtained from `read.pdb`.
`exepath`	path to the 'STRIDE' program on your system (i.e. the directory where 'STRIDE' is stored).

Details

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

Value

Returns a list with the following components:

`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".
`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.

Note

A system call is made to the 'STRIDE' 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.

'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://www.embl-heidelberg.de/stride/stride_info.html, or copy it from an installation of VMD.

See Also

read.pdb, dssp, torsion.pdb, torsion.xyz

Examples

```
## Not run:
# Read a PDB file
pdb <- read.pdb(system.file("examples/d1bg2__.ent", package="bio3d"))
sse <- stride(pdb)

# 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)
```

---

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( system.file("examples/1bg2.pdb", package="bio3d") )
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"))
pdb.path=paste(system.file(package="bio3d"),"/examples/",sep="")
m <- read.fasta.pdb(aln, pdb.path = pdb.path, pdbext = ".ent")
a <- torsion.xyz(m$xyz[2,],1)
b <- torsion.xyz(m$xyz[3,],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( system.file("examples/1bg2.pdb", package="bio3d") )
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], increment=3)

## End(Not run)

##-- Compare two PDBs to highlight interesting residues
aln <- read.fasta(system.file("examples/kif1a.fa",package="bio3d"))
pdb.path=paste(system.file(package="bio3d"),"/examples/",sep="")
m <- read.fasta.pdb(aln, pdb.path = pdb.path, pdbext = ".ent")
a <- torsion.xyz(m$xyz[2,],1)
b <- torsion.xyz(m$xyz[3,],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)
```

Arguments

pdb	a PDB structure object obtained from read.pdb.
inds	a list object of ATOM and XYZ indices as obtained from atom.select.

Details

This is a basic utility function for selecting a subset of a PDB object.

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 (version2.2) see:
http://www.rcsb.org/pdb/file_formats/pdb/pdbguide2.2/guide2.2_frame.html .

See Also

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

Examples

```
## pdb <- read.pdb( get.pdb("5p21", URLonly=TRUE) )
## npdb <- trim.pdb(pdb, inds=atom.select(pdb, "back"))
## write.pdb(npdb, file="backboneonly.pdb")
```

---

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))
```

wiki.tbl *Wiki Table*

Description

Print a matrix as a Wiki formatted table.

Usage

```
wiki.tbl(mat)
```

Arguments

mat	a matrix for printing.

Details

This function prints a basic wiki formatted table from a given matrix 'mat'.

Value

Called for its effect.

Author(s)

Barry Grant

References

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

Examples

```
data(USArrests)
wiki.tbl(USArrests)
# paste output into your PBWIKI
```

wrap.tor *Wrap Torsion Angle Data*

Description

Adjust angular data so that the absolute difference of any of the observations from its mean is not greater than 180 degrees.

Usage

```
wrap.tor(data, wrapav=TRUE, avestruc=NULL)
```

Arguments

data	a numeric vector or matrix of torsion angle data as obtained from torsion.xyz.
wrapav	logical, if TRUE average structure is also 'wrapped'
avestruc	a numeric vector corresponding to the average structure

Details

This is a basic utility function for coping with the periodicity of torsion angle data, by 'wraping' angular data such that the absolute difference of any of the observations from its column-wise mean is not greater than 180 degrees.

Value

A numeric vector or matrix of wrapped torsion angle data.

Author(s)

Karim ElSawy

References

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

See Also

torsion.xyz

---

write.crd *Write CRD File*

Description

Write a CHARMM CARD (CRD) coordinate file.

Usage

```
write.crd(pdb = NULL, xyz = pdb$xyz, resno = NULL, resid = NULL, eleno = NULL, elety = NULL, seg
```

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.charmm.org/document/Charmm/c32b2/io.html.

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( system.file("examples/1bg2.pdb", package="bio3d") )
pdb.summary(pdb)
# Convert to CHARMM format
new <- convert.pdb(pdb, type="charmm")
pdb.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.ebi.ac.uk/help/formats_frame.html.

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(system.file("examples/1bg2.pdb", package="bio3d"))

## 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/kinesin_xray.fa",package="bio3d"))

## Cut all but positions 330 to 345
aln$ali=aln$ali[,330:345]
```

```
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")
```

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.

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://amber.scripps.edu/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, 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, end = TRUE, 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.
end	logical, if TRUE TER and END lines are written.
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 (version2.2) see:
http://www.rcsb.org/pdb/file_formats/pdb/.

See Also

read.pdb, read.dcd, read.fasta.pdb, read.fasta

Examples

```
# Read a PDB file
pdb <- read.pdb(system.file("examples/d1bg2__.ent", package="bio3d"))

# 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 (version2.2) see:
http://www.rcsb.org/pdb/file_formats/pdb/.

See Also

read.pdb, read.dcd, read.fasta.pdb, read.fasta

Examples

```
# Read a PDB file
pdb <- read.pdb(system.file("examples/d1bg2__.ent", package="bio3d"))

# 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