

# Package ‘bio3d’

October 27, 2022

**Title** Biological Structure Analysis

**Version** 2.4-4

**Author** Barry Grant [aut, cre],  
Xin-Qiu Yao [aut],  
Lars Skjaerven [aut],  
Julien Ide [aut]

**VignetteBuilder** knitr

**LinkingTo** Rcpp

**Imports** Rcpp, parallel, grid, graphics, grDevices, stats, utils

**Suggests** XML, RCurl, lattice, ncd4, igrph, bigmemory, knitr,  
rmarkdown, testthat (>= 0.9.1), httr, msa, Biostrings

**Depends** R (>= 3.1.0)

**LazyData** yes

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

**Maintainer** Barry Grant <bjgrant@ucsd.edu>

**License** GPL (>= 2)

**URL** <http://thegrantlab.org/bio3d/>,  
<https://bitbucket.org/Grantlab/bio3d/>

**RoxygenNote** 7.1.1

**NeedsCompilation** yes

**Repository** CRAN

**Date/Publication** 2022-10-26 22:45:06 UTC

**R topics documented:**

bio3d-package . . . . .	6
aa.index . . . . .	7
aa.table . . . . .	9
aa123 . . . . .	10
aa2index . . . . .	11
aa2mass . . . . .	12
aanma . . . . .	14
aanma.pdb . . . . .	17
aln2html . . . . .	19
angle.xyz . . . . .	21
as.fasta . . . . .	22
as.pdb . . . . .	23
as.select . . . . .	26
atom.index . . . . .	27
atom.select . . . . .	28
atom2ele . . . . .	31
atom2mass . . . . .	33
atom2xyz . . . . .	34
basename.pdb . . . . .	35
bhattacharyya . . . . .	36
binding.site . . . . .	38
biunit . . . . .	40
blast.pdb . . . . .	41
bounds . . . . .	44
bounds.sse . . . . .	45
bwr.colors . . . . .	46
cat.pdb . . . . .	47
chain.pdb . . . . .	48
check.utility . . . . .	50
clean.pdb . . . . .	50
cmap . . . . .	52
cna . . . . .	54
cnapath . . . . .	57
com . . . . .	60
combine.select . . . . .	62
community.aln . . . . .	64
community.tree . . . . .	66
consensus . . . . .	68
conserv . . . . .	69
convert.pdb . . . . .	71
core.cmap . . . . .	73
core.find . . . . .	74
cov.nma . . . . .	78
covsoverlap . . . . .	79
dccm . . . . .	80
dccm.enma . . . . .	81

dccm.gnm . . . . .	82
dccm.nma . . . . .	84
dccm.pca . . . . .	85
dccm.xyz . . . . .	87
deformation.nma . . . . .	89
diag.ind . . . . .	91
difference.vector . . . . .	92
dist.xyz . . . . .	93
dm . . . . .	94
dssp . . . . .	97
elements . . . . .	100
entropy . . . . .	101
example.data . . . . .	104
filter.cmap . . . . .	105
filter.dccm . . . . .	106
filter.identity . . . . .	108
filter.rmsd . . . . .	110
fit.xyz . . . . .	111
fluct.nma . . . . .	114
formula2mass . . . . .	115
gap.inspect . . . . .	116
geostas . . . . .	118
get.pdb . . . . .	122
get.seq . . . . .	124
gnm . . . . .	125
hclustplot . . . . .	127
hmmmer . . . . .	129
identify.cna . . . . .	132
inner.prod . . . . .	133
inspect.connectivity . . . . .	135
is.gap . . . . .	136
is.mol2 . . . . .	137
is.pdb . . . . .	138
is.select . . . . .	139
is.xyz . . . . .	140
layout.cna . . . . .	140
lbio3d . . . . .	142
load.enmff . . . . .	142
mask . . . . .	145
mktrj . . . . .	146
motif.find . . . . .	148
mustang . . . . .	149
network.amendment . . . . .	151
nma . . . . .	153
nma.pdb . . . . .	154
nma.pdb . . . . .	158
normalize.vector . . . . .	161
orient.pdb . . . . .	162

overlap	163
pairwise	165
pca	166
pca.array	167
pca.pdbs	168
pca.tor	169
pca.xyz	171
pdb.annotate	173
pdb2aln	175
pdb2aln.ind	177
pdb2sse	179
pdbaln	180
pdffit	182
pdbs2pdb	183
pdbs2sse	185
pdbseq	186
pdbsplit	187
pfam	189
plot.bio3d	191
plot.cmap	194
plot.cna	196
plot.core	198
plot.dccm	200
plot.dmat	203
plot.enma	205
plot.fasta	207
plot.fluct	208
plot.geostas	210
plot.hmm	211
plot.matrix.loadings	213
plot.nma	214
plot.pca	216
plot.pca.loadings	218
plot.rmsip	219
print.cna	220
print.core	222
print.fasta	223
print.xyz	224
project.pca	225
prune.cna	226
pymol	228
read.all	231
read.cif	233
read.crd	235
read.crd.amber	236
read.crd.charmm	237
read.dcd	239
read.fasta	241

read.fasta.pdb . . . . .	242
read.mol2 . . . . .	244
read.ncdf . . . . .	247
read.pdb . . . . .	249
read.pdcBD . . . . .	252
read.pqr . . . . .	254
read.prmtop . . . . .	257
rgyr . . . . .	259
rle2 . . . . .	260
rmsd . . . . .	261
rmsf . . . . .	263
rmsip . . . . .	264
sdENM . . . . .	266
seq2aln . . . . .	267
seqaln . . . . .	269
seqaln.pair . . . . .	272
seqbind . . . . .	273
seqidentity . . . . .	275
setup.ncore . . . . .	276
sip . . . . .	277
sse.bridges . . . . .	278
store.atom . . . . .	279
struct.aln . . . . .	280
torsion.pdb . . . . .	282
torsion.xyz . . . . .	284
trim . . . . .	286
trim.mol2 . . . . .	288
trim.pdbs . . . . .	289
trim.xyz . . . . .	290
unbound . . . . .	291
uniprot . . . . .	292
var.xyz . . . . .	294
vec2resno . . . . .	295
vmd . . . . .	296
vmd_colors . . . . .	298
wrap.tor . . . . .	299
write.crd . . . . .	300
write.fasta . . . . .	301
write.mol2 . . . . .	303
write.ncdf . . . . .	304
write.pdb . . . . .	305
write.pir . . . . .	307
write.pqr . . . . .	309

---

bio3d-package

*Biological Structure Analysis*

---

## Description

Utilities for the analysis of protein structure and sequence data.

## Details

Package: bio3d  
Type: Package  
Version: 2.4-4  
Date: 2022-10-20  
License: GPL version 2 or newer  
URL: <http://thegrantlab.org/bio3d/>

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

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

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

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

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

## Note

The latest version, package vignettes and documentation with worked example outputs can be obtained from the bio3d website:

<http://thegrantlab.org/bio3d/>.

<http://thegrantlab.org/bio3d/reference/>.

<https://bitbucket.org/Grantlab/bio3d/>.

**Author(s)**

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

**References**

Grant, B.J. et al. (2006) *Bioinformatics* **22**, 2695–2696. Skjaerven, L. et al. (2014) *BMC Bioinformatics* **15**, 399. Grant, B.J. et al. (2021) *Protein Science* **30**, 20–30.

**Examples**

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

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

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

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

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

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

## See package vignettes and tutorials online:
##  http://thegrantlab.org/bio3d/articles/

## End(Not run)
```

---

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:

<https://www.genome.jp/aaindex/>

For a description of the 'AAindex' database see:

[https://www.genome.jp/aaindex/aaindex\\_help.html](https://www.genome.jp/aaindex/aaindex_help.html).

## References

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

'AAIndex' is the work of Kanehisa and co-workers:

Kawashima and Kanehisa (2000) *Nucleic Acids Res.* **28**, 374;

Tomii and Kanehisa (1996) *Protein Eng.* **9**, 27–36;

Nakai, Kidera and Kanehisa (1988) *Protein Eng.* **2**, 93–100.

## Examples

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

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

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

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

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

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



---

`aa.table`*Table of Relevant Amino Acids*

---

**Description**

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

**Usage**

```
aa.table
```

**Format**

A data frame with the following components.

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

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

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

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

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

**References**

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

**See Also**

[aa2mass](#), [aa.index](#), [atom.index](#), [elements](#),

**Examples**

```
data(aa.table)
aa.table

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

## read PDB, and fetch residue masses
pdb <- read.pdb(system.file("examples/1hel.pdb", package="bio3d"))
aa2mass(pdb)
```

---

aa123

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

---

### Description

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

### Usage

aa123(aa)

aa321(aa)

### Arguments

aa                    a character vector of individual aminoacid codes.

### Details

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

### Value

A character vector of aminoacid codes.

### Author(s)

Barry Grant

### References

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

For a description of IUPAC one-letter codes see:

[https://www.insdc.org/documents/feature\\_table.html#7.4.3](https://www.insdc.org/documents/feature_table.html#7.4.3)

For more information on PDB residue codes see:

<http://ligand-expo.rcsb.org/ld-search.html>

### See Also

[read.pdb](#), [read.fasta](#), [pdbseq](#)

**Examples**

```

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

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

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

# Alternative approach for ATOM sequence extraction
pdbseq(pdb)
pdbseq(pdb, aa1=FALSE )

## 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:

<https://www.genome.jp/aaindex/> or the [aa.index](#) documentation.

**See Also**

[aa.index](#), [read.fasta](#)

**Examples**

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

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

## Use an alignment

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

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

## End(Not run)
```

---

aa2mass

*Amino Acid Residues to Mass Converter*

---

**Description**

Convert a sequence of amino acid residue names to mass.

**Usage**

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

## Arguments

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

## Details

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

See examples for more details.

## Value

Returns a numeric vector of masses.

## Note

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

## Author(s)

Lars Skjaerven

## References

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

## See Also

[atom.index](#), [atom2mass](#), [aa.index](#)

## Examples

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

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

## or
```

```

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

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

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

## End(Not run)

```

---

aanma

*All Atom Normal Mode Analysis*


---

## Description

Perform all-atom elastic network model normal modes calculation of a protein structure.

## Usage

```

aanma(...)

## S3 method for class 'pdb'
aanma(pdb, pfc.fun = NULL, mass = TRUE, temp = 300,
      keep = NULL, hessian = NULL, outmodes = "calpha", rm.wat = TRUE,
      reduced = FALSE, rtb = FALSE, nmer = 1, ...)

rtb(hessian, pdb, mass = TRUE, nmer = 1, verbose = TRUE)

```

## Arguments

...	additional arguments to <a href="#">build.hessian</a> and <a href="#">aa2mass</a> . One useful option here for dealing with unconventional residues is 'mass.custom', see the <a href="#">aa2mass</a> function for details.
pdb	an object of class <code>pdb</code> as obtained from function <a href="#">read.pdb</a> .
pfc.fun	customized pair force constant ('pfc') function. The provided function should take a vector of distances as an argument to return a vector of force constants. If <code>NULL</code> , the default function 'aaenm2' will be employed. (See details below).
mass	logical, if <code>TRUE</code> the Hessian will be mass-weighted.
temp	numerical, temperature for which the amplitudes for scaling the atomic displacement vectors are calculated. Set 'temp=NULL' to avoid scaling.
keep	numerical, final number of modes to be stored. Note that all subsequent analyses are limited to this subset of modes. This option is useful for very large structures and cases where memory may be limited.
hessian	hessian matrix as obtained from <a href="#">build.hessian</a> . For internal purposes and generally not intended for public use.

outmodes	either a character ('calpha' or 'noh') or atom indices as obtained from <a href="#">atom.select</a> specifying the atoms to include in the resulting mode object. (See details below).
rm.wat	logical, if TRUE water molecules will be removed before calculation.
reduced	logical, if TRUE the coarse-grained ('4-bead') ENM will be employed. (See details below).
rtb	logical, if TRUE the rotation-translation block based approximate modes will be calculated. (See details below).
nmer	numerical, defines the number of residues per block (used only when <code>rtb=TRUE</code> ).
verbose	logical, if TRUE print detailed processing message

### Details

This function builds an elastic network model (ENM) based on all heavy atoms of input `pdb`, and performs subsequent normal mode analysis (NMA) in various manners. By default, the 'aaenm2' force field (defining of the spring constants between atoms) is used, which was obtained by fitting to a local energy minimum of a crambin model derived from the AMBER99SB force field. It employs a pair force constant function which falls as  $r^{-6}$ , and specific force constants for covalent and intra-residue atom pairs. See also [load.enmff](#) for other force field options.

The `outmodes` argument controls the type of output modes. There are two standard types of output modes: 'noh' and 'calpha'. `outmodes='noh'` invokes regular all-atom based ENM-NMA. When `outmodes='calpha'`, an effective Hessian with respect to all C-alpha atoms will be first calculated using the same formula as in Hinsen et al. NMA is then performed on this effective C-alpha based Hessian. In addition, users can provide their own atom selection (see [atom.select](#)) as the value of `outmodes` for customized output modes generation.

When `reduced=TRUE`, only a selection of all heavy atoms is used to build the ENM. More specifically, three to five atoms per residue constitute the model. Here the N, CA, C atoms represent the protein backbone, and zero to two selected side chain atoms represent the side chain (selected based on side chain size and the distance to CA). This coarse-grained ENM has significantly improved computational efficiency and similar prediction accuracy with respect to the all-atom ENM.

When `rtb=TRUE`, rotation-translation block (RTB) based approximate modes will be calculated. In this method, each residue is assumed to be a rigid body (or 'block') that has only rotational and translational degrees of freedom. Intra-residue deformation is thus ignored. (See Durand et al 1994 and Tama et al. 2000 for more details). N residues per block is also supported, where  $N=1, 2, 3$ , etc. (See argument `nmer`). The RTB method has significantly improved computational efficiency and similar prediction accuracy with respect to the all-atom ENM.

By default the function will diagonalize the mass-weighted Hessian matrix. The resulting mode vectors are moreover scaled by the thermal fluctuation amplitudes.

### Value

Returns an object of class 'nma' with the following components:

modes	numeric matrix with columns containing the normal mode vectors. Mode vectors are converted to unweighted Cartesian coordinates when <code>mass=TRUE</code> . Note that the 6 first trivial eigenvectors appear in columns one to six.
frequencies	numeric vector containing the vibrational frequencies corresponding to each mode (for <code>mass=TRUE</code> ).

force.constants	numeric vector containing the force constants corresponding to each mode (for mass=FALSE)).
fluctuations	numeric vector of atomic fluctuations.
U	numeric matrix with columns containing the raw eigenvectors. Equals to the modes component when mass=FALSE and temp=NULL.
L	numeric vector containing the raw eigenvalues.
xyz	numeric matrix of class xyz containing the Cartesian coordinates in which the calculation was performed.
mass	numeric vector containing the residue masses used for the mass-weighting.
temp	numerical, temperature for which the amplitudes for scaling the atomic displacement vectors are calculated.
triv.modes	number of trivial modes.
natoms	number of C-alpha atoms.
call	the matched call.

### Author(s)

Lars Skjaerven & Xin-Qiu Yao

### References

Hinsen, K. et al. (2000) *Chem. Phys.* **261**, 25. Durand, P. et al. (1994) *Biopolymers* **34**, 759. Tama, F. et al. (2000) *Proteins* **41**, 1.

### See Also

[nma.pdb](#) for C-alpha based NMA, [aanma.pdb](#)s for ensemble all-atom NMA, [load.enmff](#) for available ENM force fields, and [fluct.nma](#), [mktrj.nma](#), and [dccm.nma](#) for various post-NMA calculations.

### Examples

```
## Not run:
# All-atom NMA takes relatively long time - Don't run by default.

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

## Calculate all-atom normal modes
modes.aa <- aanma(pdb, outmodes='noh')

## Calculate all-atom normal modes with RTB approximation
modes.aa.rtb <- aanma(pdb, outmodes='noh', rtb=TRUE)

## Compare the two modes
rmsip(modes.aa, modes.aa.rtb)
```



```

## Calculate C-alpha normal modes.
modes <- aanma(pdb)

## Calculate C-alpha normal modes with reduced ENM.
modes.cg <- aanma(pdb, reduced=TRUE)

## Calculate C-alpha normal modes with RTB approximation
modes.rtb <- aanma(pdb, rtb=TRUE)

## Compare modes
rmsip(modes, modes.cg)
rmsip(modes, modes.rtb)

## Print modes
print(modes)

## Plot modes
plot(modes)

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

## End(Not run)

```

---

aanma.pdbs

*Ensemble Normal Mode Analysis with All-Atom ENM*


---

## Description

Perform normal mode analysis (NMA) on an ensemble of aligned protein structures using all-atom elastic network model (aaENM).

## Usage

```

## S3 method for class 'pdbs'
aanma(pdbs, fit = TRUE, full = FALSE, subspace = NULL,
      rm.gaps = TRUE, ligand = FALSE, outpath = NULL, gc.first = TRUE,
      ncore = NULL, ...)

```

## Arguments

pdbs	an 'pdbs' object as obtained from <a href="#">read.all</a> .
fit	logical, if TRUE C-alpha coordinate based superposition is performed prior to normal mode calculations.
full	logical, if TRUE return the complete, full structure, 'nma' objects.
subspace	number of eigenvectors to store for further analysis.
rm.gaps	logical, if TRUE obtain the hessian matrices for only atoms in the aligned positions (non-gap positions in all aligned structures). Thus, gap positions are removed from output.

ligand	logical, if TRUE ligand molecules are also included in the calculation.
outpath	character string specifying the output directory to which the PDB structures should be written.
gc.first	logical, if TRUE will call gc() first before mode calculation for each structure. This is to avoid memory overload when ncore > 1.
ncore	number of CPU cores used to do the calculation.
...	additional arguments to <a href="#">aanma</a> .

### Details

This function builds elastic network model (ENM) using all heavy atoms and performs subsequent normal mode analysis (NMA) on a set of aligned protein structures obtained with function [read.all](#). The main purpose is to automate ensemble normal mode analysis using all-atom ENMs.

By default, the effective Hessian for all C-alpha atoms is calculated based on the Hessian built from all heavy atoms (including ligand atoms if `ligand=TRUE`). Returned values include aligned mode vectors and (when `full=TRUE`) a list containing the full 'nma' objects one per each structure. When '`rm.gaps=TRUE`' the unaligned atoms are omitted from output. With default arguments '`rmsip`' provides RMSIP values for all pairwise structures.

When `outmodes` is provided and is not 'alpha' (e.g. 'noh'. See [aanma](#) for more details), the function simply returns a list of 'nma' objects, one per each structure, and no aligned mode vector is returned. In this case, the arguments `full`, `subspace`, and `rm.gaps` are ignored. This is equivalent to a wrapper function repeatedly calling [aanma](#).

### Value

Returns a list of 'nma' objects (`outmodes` is provided and is not 'alpha') or an 'enma' object with the following components:

fluctuations	a numeric matrix containing aligned atomic fluctuations with one row per input structure.
rmsip	a numeric matrix of pair wise RMSIP values (only the ten lowest frequency modes are included in the calculation).
U.subspace	a three-dimensional array with aligned eigenvectors (corresponding to the subspace defined by the first N non-trivial eigenvectors ('U') of the 'nma' object).
L	numeric matrix containing the raw eigenvalues with one row per input structure.
full.nma	a list with a nma object for each input structure (available only when <code>full=TRUE</code> ).

### Author(s)

Xin-Qiu Yao & Lars Skjaerven

### See Also

For normal mode analysis on single structure PDB: [aanma](#)

For conventional C-alpha based normal mode analysis: [nma](#), [nma.pdbs](#).

For the analysis of the resulting 'eNMA' object: [mktrj.enma](#), [dccm.enma](#), [plot.enma](#), [cov.enma](#).

Similarity measures: [sip](#), [covoverlap](#), [bhattacharyya](#), [rmsip](#).

Related functionality: [read.all](#).

## Examples

```
# Needs MUSCLE installed - testing excluded
if(check.utility("muscle")) {

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

  ## Sequence Alignment
  aln <- pdbaln(files, outfile = tempfile())

  ## Read all pdb coordinates
  pdbs <- read.all(aln)

  ## Normal mode analysis on aligned data
  modes <- aanma(pdbs, rm.gaps=TRUE)

  ## Plot fluctuation data
  plot(modes, pdbs=pdbs)

  ## Cluster on Fluctuation similarity
  sip <- sip(modes)
  hc <- hclust(dist(sip))
  col <- cutree(hc, k=3)

  ## Plot fluctuation data
  plot(modes, pdbs=pdbs, col=col)

  ## RMSIP is pre-calculated
  heatmap(1-modes$rmsip)

  ## Bhattacharyya coefficient
  bc <- bhattacharyya(modes)
  heatmap(1-bc)

}
```

## Description

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

**Usage**

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

**Arguments**

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

**Value**

Called for its effect.

**Note**

Your web browser should support style sheets.

**Author(s)**

Barry Grant

**References**

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

**See Also**

[read.fasta](#), [write.fasta](#), [seqaln](#)

## Examples

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

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

## End(Not run)
```

---

angle.xyz

*Calculate the Angle Between Three Atoms*

---

## Description

A function for basic bond angle determination.

## Usage

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

## Arguments

xyz	a numeric vector of Cartesian 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 successive 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 successive 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( system.file("examples/1hel.pdb", package="bio3d") )

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

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

---

as.fasta

*Alignment to FASTA object*


---

**Description**

Convert alignment/sequence in matrix/vector format to FASTA object.

**Usage**

```
as.fasta(x, id=NULL, ...)
```

**Arguments**

x	a sequence character matrix/vector (e.g obtained from <a href="#">get.seq</a> or <a href="#">seqbind</a> ).
id	a vector of sequence names to serve as sequence identifiers. By default the function will use the row names of the alignment if they exist, otherwise ids will be generated.
...	arguments passed to and from functions.

**Details**

This function provides basic functionality to convert a sequence character matrix/vector to a FASTA object.

**Value**

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

ali	an alignment character matrix with a row per sequence and a column per equivalent aminoacid/nucleotide.
id	sequence names as identifiers.
call	the matched call.

**Author(s)**

Lars Skjaerven

**References**Grant, B.J. et al. (2006) *Bioinformatics* **22**, 2695–2696.**See Also**[get.seq](#), [seqaln](#), [seqbind](#), [pdbaln](#)**Examples**

```
as.fasta(c("A", "C", "D"))
```

---

`as.pdb`*Convert to PDB format*

---

**Description**

Convert Tripos Mol2 format, or Amber parameter/topology and coordinate data to PDB format.

**Usage**

```
as.pdb(...)

## S3 method for class 'mol2'
as.pdb(mol, ...)

## S3 method for class 'prmtop'
as.pdb(prmtop, crd=NULL, inds=NULL, inds.crd=inds, ncore=NULL, ...)

## Default S3 method:
as.pdb(pdb=NULL, xyz=NULL, type=NULL, resno=NULL,
       resid=NULL, eleno=NULL, elety=NULL, chain=NULL,
       insert=NULL, alt=NULL, o=NULL, b=NULL, segid=NULL,
       elesy=NULL, charge=NULL, verbose=TRUE, ...)
```

**Arguments**

<code>...</code>	arguments passed to and from functions.
<code>mol</code>	a list object of type "mol2" (obtained with <a href="#">read.mol2</a> ).
<code>prmtop</code>	a list object of type "prmtop" (obtained with <a href="#">read.prmtop</a> ).
<code>crd</code>	a list object of type "crd" (obtained with <a href="#">read.crd.amber</a> ).
<code>inds</code>	a list object of type "select" as obtained from <a href="#">atom.select</a> . The indices points to which atoms in the PRMTOP object to convert.

inds.crd	same as the 'inds' argument, but pointing to the atoms in CRD object to convert. By default, this argument equals to 'inds', assuming the same number and sequence of atoms in the PRMTOP and CRD objects.
ncore	number of CPU cores used to do the calculation. <code>ncore&gt;1</code> requires package 'parallel' installed.
pdb	an object of class 'pdb' as obtained from <a href="#">read.pdb</a> .
xyz	a numeric vector/matrix of Cartesian coordinates. If provided, the number of atoms in the new PDB object will be set to <code>ncol(as.xyz(xyz))/3</code> (see <a href="#">as.xyz</a> ). If xyz is not provided the number of atoms will be based on the length of <code>e1eno</code> , <code>resno</code> , or <code>resid</code> (in that order).
type	a character vector of record types, i.e. "ATOM" or "HETATM", with length equal to <code>ncol(as.xyz(xyz))/3</code> . Alternatively, a single element character vector can be provided which will be repeated to match the number of atoms.
resno	a numeric vector of residue numbers of length equal to <code>ncol(as.xyz(xyz))/3</code> .
resid	a character vector of residue types/ids of length equal to <code>ncol(as.xyz(xyz))/3</code> . Alternatively, a single element character vector can be provided which will be repeated to match the number of atoms.
e1eno	a numeric vector of element/atom numbers of length equal to <code>ncol(as.xyz(xyz))/3</code> .
e1ety	a character vector of element/atom types of length equal to <code>ncol(as.xyz(xyz))/3</code> . Alternatively, a single element character vector can be provided which will be repeated to match the number of atoms.
chain	a character vector of chain identifiers with length equal to <code>ncol(as.xyz(xyz))/3</code> . Alternatively, a single element character vector can be provided which will be repeated to match the number of atoms.
insert	a character vector of insertion code with length equal to <code>ncol(as.xyz(xyz))/3</code> .
alt	a character vector of alternate record with length equal to <code>ncol(as.xyz(xyz))/3</code> .
o	a numeric vector of occupancy values of length equal to <code>ncol(as.xyz(xyz))/3</code> . Alternatively, a single element numeric vector can be provided which will be repeated for to match the number of atoms.
b	a numeric vector of B-factors of length equal to <code>ncol(as.xyz(xyz))/3</code> . Alternatively, a single element numeric vector can be provided which will be repeated to match the number of atoms.
segid	a character vector of segment id of length equal to <code>ncol(as.xyz(xyz))/3</code> . Alternatively, a single element character vector can be provided which will be repeated to match the number of atoms.
e1esy	a character vector of element symbol of length equal to <code>ncol(as.xyz(xyz))/3</code> . Alternatively, a single element character vector can be provided which will be repeated to match the number of atoms.
charge	a numeric vector of atomic charge of length equal to <code>ncol(as.xyz(xyz))/3</code> .
verbose	logical, if TRUE details of the PDB generation process is printed to screen.



## Details

This function converts Tripos Mol2 format, Amber formatted parameter/topology (PRMTOP) and coordinate objects, and vector data to a PDB object.

While `as.pdb.mol2` and `as.pdb.prmtop` converts specific objects to a PDB object, `as.pdb.default` provides basic functionality to convert raw data such as vectors of e.g. residue numbers, residue identifiers, Cartesian coordinates, etc to a PDB object. When `pdb` is provided the returned PDB object is built from the input object with fields replaced by any input vector arguments. e.g. `as.pdb(pdb, xyz=crd)` will return the same PDB object, with only the Cartesian coordinates changed to `crd`.

## Value

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

<code>atom</code>	a data.frame 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).
<code>xyz</code>	a numeric matrix of ATOM coordinate data of class xyz.
<code>calpha</code>	logical vector with length equal to <code>nrow(atom)</code> with TRUE values indicating a C-alpha "eley".
<code>call</code>	the matched call.

## Author(s)

Lars Skjaerven

## References

Grant, B.J. et al. (2006) *Bioinformatics* **22**, 2695–2696. <http://ambermd.org/FileFormats.php>

## See Also

[read.crd](#), [read.ncdf](#), [atom.select](#), [read.pdb](#)

## Examples

```
## Vector(s) to PDB object
pdb <- as.pdb(resno=1:6, eley="CA", resid="ALA", chain="A")
pdb

## Not run:
## Read a PRMTOP file
prmtop <- read.prmtop(system.file("examples/crambin.prmtop", package="bio3d"))

## Read Amber coordinates
crds <- read.crd(system.file("examples/crambin.inpcrd", package="bio3d"))

## Atom selection
ca.indes <- atom.select(prmtop, "calpha")
```

```
## Convert to PDB format
pdb <- as.pdb(prmtop, crds, inds=ca.inds)

## Read a single entry MOL2 file
## (returns a single object)
mol <- read.mol2( system.file("examples/aspirin.mol2", package="bio3d") )

## Convert to PDB
pdb <- as.pdb(mol)

## End(Not run)
```

---

as.select

*Convert Atomic Indices to a Select Object*

---

### Description

Convert atomic indices to a select object with 'atom' and 'xyz' components.

### Usage

```
as.select(x, ...)
```

### Arguments

x	a numeric vector containing atomic indices to be converted to a 'select' object. Alternatively, a logical vector can be provided.
...	arguments passed to and from functions.

### Details

Convert atomic indices to a select object with 'atom' and 'xyz' components.

### Value

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

atom	a numeric matrix of atomic indices.
xyz	a numeric matrix of xyz indices.
call	the matched call.

### Author(s)

Lars Skjaerven

**References**

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

**See Also**

[atom.select](#), [read.pdb](#)

**Examples**

```
as.select(c(1,2,3))
```

---

atom.index

*Atom Names/Types*

---

**Description**

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

**Usage**

```
atom.index
```

**Format**

A data frame with the following components.

name a character vector containing atom names/types.

symb a character vector containing atomic symbols.

**References**

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

**See Also**

[elements](#), [atom.index](#), [atom2ele](#)

**Examples**

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

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

atom.select

*Atom Selection from PDB and PRMTOP Structure Objects***Description**

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

**Usage**

```
atom.select(...)

## S3 method for class 'pdb'
atom.select(pdb, string = NULL,
            type = NULL, eleno = NULL, elety = NULL,
            resid = NULL, chain = NULL, resno = NULL,
            insert = NULL, segid = NULL,
            operator = "AND", inverse = FALSE,
            value = FALSE, verbose=FALSE, ...)

## S3 method for class 'pdbs'
atom.select(pdbs, string = NULL,
            resno = NULL, chain = NULL, resid = NULL,
            operator="AND", inverse = FALSE,
            value = FALSE, verbose=FALSE, ...)

## S3 method for class 'mol2'
atom.select(mol, string=NULL,
            eleno = NULL, elena = NULL, elety = NULL,
            resid = NULL, chain = NULL, resno = NULL,
            statbit = NULL,
            operator = "AND", inverse = FALSE,
            value = FALSE, verbose=FALSE, ...)

## S3 method for class 'prmtop'
atom.select(prmtop, ...)

## S3 method for class 'select'
print(x, ...)
```

**Arguments**

...	arguments passed to atom.select.pdb, atom.select.prmtop, or print.
pdb	a structure object of class "pdb", obtained from <a href="#">read.pdb</a> .
pdbs	a numeric matrix of aligned C-alpha xyz Cartesian coordinates as obtained with <a href="#">read.fasta.pdb</a> or <a href="#">pdbaln</a> .
string	a single selection keyword from calpha cbeta backbone sidechain protein nucleic ligand water h or noh.

type	a single element character vector for selecting 'ATOM' or 'HETATM' record types.
eleno	a numeric vector of element numbers.
elena	a character vector of atom names.
elety	a character vector of atom names.
resid	a character vector of residue name identifiers.
chain	a character vector of chain identifiers.
resno	a numeric vector of residue numbers.
insert	a character vector of insert identifiers. Non-insert residues can be selected with NA or '' values. The default value of NULL will select both insert and non-insert residues.
segid	a character vector of segment identifiers. Empty segid values can be selected with NA or '' values. The default value of NULL will select both empty and non-empty segment identifiers.
operator	a single element character specifying either the AND or OR operator by which individual selection components should be combined. Allowed values are "AND" and "OR".
verbose	logical, if TRUE details of the selection are printed.
inverse	logical, if TRUE the inversed selection is returned (i.e. all atoms NOT in the selection).
value	logical, if FALSE, vectors containing the (integer) indices of the matches determined by atom.select are returned, and if TRUE, a pdb object containing the matching atoms themselves is returned.
mol	a structure object of class "mol2", obtained from <a href="#">read.mol2</a> .
statbit	a character vector of statbit identifiers.
prmtop	a structure object of class "prmtop", obtained from <a href="#">read.prmtop</a> .
x	a atom.select object as obtained from <a href="#">atom.select</a> .

## Details

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

Input selection criteria include selection string keywords (such as "calpha", "backbone", "sidechain", "protein", "nucleic", "ligand", etc.) and individual named selection components (including 'chain', 'resno', 'resid', 'elety' etc.).

For example, `atom.select(pdb, "calpha")` will return indices for all C-alpha (CA) atoms found in protein residues in the pdb object, `atom.select(pdb, "backbone")` will return indices for all protein N,CA,C,O atoms, and `atom.select(pdb, "cbeta")` for all protein N,CA,C,O,CB atoms.

Note that keyword string shortcuts can be combined with individual selection components, e.g. `atom.select(pdb, "protein", chain="A")` will select all protein atoms found in chain A.

Selection criteria are combined according to the provided operator argument. The default operator AND (or &) will combine by intersection while OR (or |) will take the union.

For example, `atom.select(pdb, "protein", eley=c("N", "CA", "C"), resno=65:103)` will select the N, CA, C atoms in the protein residues 65 through 103, while `atom.select(pdb, "protein", resid="ATP", operator="OR")` will select all protein atoms as well as any ATP residue(s).

Other string shortcuts include: "calpha", "back", "backbone", "cbeta", "protein", "notprotein", "ligand", "water", "notwater", "h", "noh", "nucleic", and "notnucleic".

In addition, the `combine.select` function can further combine atom selections using 'AND', 'OR', or 'NOT' logical operations.

### Value

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

atom	a numeric matrix of atomic indices.
xyz	a numeric matrix of xyz indices.
call	the matched call.

### Note

Protein atoms are defined as any atom in a residue matching the residue name in the attached `aa.table` data frame. See `aa.table$aa3` for a complete list of residue names.

Nucleic atoms are defined as all atoms found in residues with names A, U, G, C, T, I, DA, DU, DG, DC, DT, or DI.

Water atoms/residues are defined as those with residue names H2O, OH2, HOH, HHO, OHH, SOL, WAT, TIP, TIP3, or TIP4.

### Author(s)

Barry Grant, Lars Skjaerven

### References

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

### See Also

[read.pdb](#), [as.select](#), [combine.select](#), [trim.pdb](#), [write.pdb](#), [read.prmtop](#), [read.crd](#), [read.dcd](#), [read.ncdf](#).

### Examples

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

# Select protein atoms of chain A
atom.select(pdb, "protein", chain="A")
```

```
# Select all atoms except from the protein
atom.select(pdb, "protein", inverse=TRUE, verbose=TRUE)

# Select all C-alpha atoms with residues numbers between 43 and 54
sele <- atom.select(pdb, "calpha", resno=43:54, verbose=TRUE)

# Access the PDB data with the selection indices
print( pdb$atom[ sele$atom, "resid" ] )
print( pdb$xyz[ sele$xyz ] )

# Trim PDB to selection
ca.pdb <- trim.pdb(pdb, sele)

## Not run:

##- PRMTOP example
prmtop <- read.prmtop(system.file("examples/crambin.prmtop", package="bio3d"))

## Atom selection
ca.inds <- atom.select(prmtop, "calpha")

## End(Not run)
```

---

atom2ele

*Atom Names/Types to Atomic Symbols Converter*

---

## Description

Convert atom names/types into atomic symbols

## Usage

```
atom2ele(...)
```

## Default S3 method:

```
atom2ele(x, eley.custom=NULL, rescue=TRUE, ...)
```

## S3 method for class 'pdb'

```
atom2ele(pdb, inds=NULL, ...)
```

## Arguments

x	a character vector containing atom names/types to be converted.
eley.custom	a customized data.frame containing atom names/types and corresponding atomic symbols.
rescue	logical, if TRUE the atomic symbols will be converted based on matching with <code>bio3d::elements\$symb</code> .

pdb            an object of class 'pdb' for which eley will be converted.  
inds           an object of class 'select' indicating a subset of the pdb object to be used (see [atom.select](#) and [trim.pdb](#)).  
...            further arguments passed to or from other methods.

### Details

The default method searches for the atom names/types in the [atom.index](#) data set and returns their corresponding atomic symbols. If `eley.custom` is specified it is combined with [atom.index](#) (using `rbind`) before searching. Therefore, `eley.custom` must contain columns named `name` and `symp`.

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

### Value

Return a character vector of atomic symbols

### Author(s)

Julien Ide, Lars Skjaerven

### See Also

[atom.index](#), [elements](#), [read.pdb](#), [atom2mass](#), [formula2mass](#)

### Examples

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

# PDB server connection required - testing excluded

## Get atomic symbols from a PDB object with a customized data set
pdb <- read.pdb("3RE0", verbose=FALSE)
lig <- trim(pdb, "ligand")

## maps PT1 to Pt, CL2 to Cl, C4A to C
atom2ele(lig)

## map atom name to element manually
myeley <- data.frame(name = "CL2", symb = "Cl")
atom2ele(lig, eley.custom = myeley)
```



---

`atom2mass`*Atom Names/Types to Mass Converter*

---

## Description

Convert atom names/types into atomic masses.

## Usage

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

## Arguments

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

## Details

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

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

## Value

Return a numeric vector of masses.

**Author(s)**

Julien Ide, Lars Skjaerven

**See Also**

[elements](#), [atom.index](#), [atom2ele](#), [read.pdb](#)

**Examples**

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

# PDB server connection required - testing excluded

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

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

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

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

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

---

atom2xyz

*Convert Between Atom and xyz Indices*

---

**Description**

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

**Usage**

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

### Arguments

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

### Value

A numeric vector of either xyz or atom indices.

### Author(s)

Barry Grant

### References

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

### See Also

[atom.select](#), [read.pdb](#)

### Examples

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

---

basename.pdb

*Manipulate PDB File Names*

---

### Description

Removes all of the path up to and including the last path separator (if any) and the final ‘.pdb’ extension.

### Usage

```
basename.pdb(x, mk4 = FALSE, ext=".pdb")
```

### Arguments

x            character vector of PDB file names, containing path and extensions.  
 mk4        logical, if TRUE the output will be truncated to the first 4 characters of the  
 basename. This is frequently convenient for matching RCSB PDB identifier  
 conventions (see examples below).  
 ext        character, specifying the file extension, e.g. ‘.pdb’ or ‘.mol2’.

**Details**

This is a simple utility function for the common task of PDB file name manipulation. It is used internally in several bio3d functions and can be thought of as `basename` for PDB files.

**Value**

A character vector of the same length as the input 'x'.

Paths not containing any separators are taken to be in the current directory.

If an element of input is 'x' is 'NA', so is the result.

**Author(s)**

Barry Grant

**References**

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

**See Also**

[basename](#), [dirname](#)

**Examples**

```
basename.pdb("/somedir/somewhere/1bg2_myfile.pdb")
basename.pdb("/somedir/somewhere/1bg2_myfile.pdb", TRUE)
```

---

bhattacharyya

*Bhattacharyya Coefficient*

---

**Description**

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

**Usage**

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

```
## S3 method for class 'nma'  
bhattacharyya(...)
```

```
## S3 method for class 'pca'  
bhattacharyya(...)
```

### Arguments

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

### Details

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

### Value

Returns the similarity coefficient(s).

### Author(s)

Lars Skjaerven

### References

Skjaerven, L. et al. (2014) *BMC Bioinformatics* **15**, 399. Grant, B.J. et al. (2006) *Bioinformatics* **22**, 2695–2696. Fuglebakk, E. et al. (2013) *JCTC* **9**, 5618–5628.

### See Also

Other similarity measures: [sip](#), [covoverlap](#), [rmsip](#).

binding.site

*Binding Site Residues***Description**

Determines the interacting residues between two PDB entities.

**Usage**

```
binding.site(a, b=NULL, a.indes=NULL, b.indes=NULL, cutoff=5,
            hydrogens=TRUE, byres=TRUE, verbose=FALSE)
```

**Arguments**

a	an object of class <code>pdb</code> as obtained from function <code>read.pdb</code> .
b	an object of class <code>pdb</code> as obtained from function <code>read.pdb</code> .
a.indes	atom and xyz coordinate indices obtained from <code>atom.select</code> that selects the elements of a upon which the calculation should be based.
b.indes	atom and xyz coordinate indices obtained from <code>atom.select</code> that selects the elements of b upon which the calculation should be based.
cutoff	distance cutoff
hydrogens	logical, if <code>FALSE</code> hydrogen atoms are omitted from the calculation.
byres	logical, if <code>TRUE</code> all atoms in a contacting residue is returned.
verbose	logical, if <code>TRUE</code> details of the selection are printed.

**Details**

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

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

If `b=a.indes=b.indes=NULL` the function will use `atom.select` with arguments "protein" and "ligand" to determine receptor and ligand, respectively.

**Value**

Returns a list with the following components:

inds	object of class <code>select</code> with <code>atom</code> and <code>xyz</code> components.
inds\$atom	atom indices of a.
inds\$xyz	xyz indices of a.
resnames	a character vector of interacting residues.
resno	a numeric vector of interacting residues numbers.
chain	a character vector of the associated chain identifiers of "resno".
call	the matched call.

**Author(s)**

Lars Skjaerven

**References**Grant, B.J. et al. (2006) *Bioinformatics* **22**, 2695–2696.**See Also**[read.pdb](#), [atom.select](#), [dm](#)**Examples**

```
# PDB server connection required - testing excluded

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

## automatically identify 'protein' and 'ligand'
bs <- binding.site(pdb)

bs$resnames
#pdb$atom[bs$inds$atom, ]

# provide indices
rec.inds <- atom.select(pdb, chain="A", resno=1:350)
lig.inds <- atom.select(pdb, chain="A", resno=351)
bs <- binding.site(pdb, a.inds=rec.inds, b.inds=lig.inds)

## Not run:
# Interaction between peptide and protein
rec.inds <- atom.select(pdb, chain='A', resno=c(1:350))
lig.inds <- atom.select(pdb, chain='I', resno=c(5:24))
bs <- binding.site(pdb, a.inds=rec.inds, b.inds=lig.inds)

## End(Not run)

# Redundant testing excluded

# Interaction between two PDB entities
#rec <- read.pdb("receptor.pdb")
#lig <- read.pdb("ligand.pdb")
rec <- trim.pdb(pdb, inds=rec.inds)
lig <- trim.pdb(pdb, inds=lig.inds)
bs <- binding.site(rec, lig, hydrogens=FALSE)
```

---

biounit *Biological Units Construction*

---

**Description**

Construct biological assemblies/units based on a 'pdb' object.

**Usage**

```
biounit(pdb, biomat = NULL, multi = FALSE, ncore = NULL)
```

**Arguments**

pdb	an object of class pdb as obtained from function <a href="#">read.pdb</a> .
biomat	a list object as returned by <code>read.pdb</code> ( <code>pdb\$remark\$biomat</code> ), containing matrices for symmetry operation on individual chains to build biological units. It will override the matrices stored in <code>pdb</code> .
multi	logical, if TRUE the biological unit is returned as a 'multi-model' pdb object with each symmetric copy a distinct structural 'MODEL'. Otherwise, all copies are represented as separated chains.
ncore	number of CPU cores used to do the calculation. By default ( <code>ncore=NULL</code> ), use all available CPU cores.

**Details**

A valid structural/simulation study should be performed on the biological unit of a protein system. For example, the alpha2-beta2 tetramer form of hemoglobin. However, canonical PDB files usually contain the asymmetric unit of the crystal cell, which can be:

1. One biological unit
2. A portion of a biological unit
3. Multiple biological units

The function performs symmetry operations to the coordinates based on the transformation matrices stored in a 'pdb' object returned by [read.pdb](#), and returns biological units stored as a list of pdb objects.

**Value**

a list of pdb objects with each representing an individual biological unit.

**Author(s)**

Xin-Qiu Yao

**See Also**

[read.pdb](#)



**Examples**

```

# PDB server connection required - testing excluded

pdb <- read.pdb("2dn1")
biounit <- biounit(pdb)
pdb
biounit

## Not run:
biounit <- biounit(read.pdb("2bfu"), multi=TRUE)
write.pdb(biounit[[1]], file="biounit.pdb")
# open the pdb file in VMD to have a look on the biological unit

## End(Not run)

```

blast.pdb

*NCBI BLAST Sequence Search and Summary Plot of Hit Statistics***Description**

Run NCBI blastp, on a given sequence, against the PDB, NR and swissprot sequence databases. Produce plots that facilitate hit selection from the match statistics of a BLAST result.

**Usage**

```

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

get.blast(urlget, time.out = NULL, chain.single=TRUE)

## S3 method for class 'blast'
plot(x, cutoff = NULL, cut.seed=NULL, cluster=TRUE, mar=c(2, 5, 1, 1), cex=1.5, ...)

```

**Arguments**

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

x	BLAST results as obtained from the function <code>blast.pdb</code> .
cutoff	A numeric cutoff value, in terms of minus the log of the evalue, for returned hits. If null then the function will try to find a suitable cutoff near 'cut.seed' which can be used as an initial guide (see below).
cut.seed	A numeric seed cutoff value, used for initial cutoff estimation. If null then a seed position is set to the point of largest drop-off in normalized scores (i.e. the biggest jump in E-values).
cluster	Logical, if TRUE (and 'cutoff' is null) a clustering of normalized scores is performed to partition hits in groups by similarity to query. If FALSE the partition point is set to the point of largest drop-off in normalized scores.
mar	A numerical vector of the form <code>c(bottom, left, top, right)</code> which gives the number of lines of margin to be specified on the four sides of the plot.
cex	a numerical single element vector giving the amount by which plot labels should be magnified relative to the default.
...	extra plotting arguments.

### Details

The `blast.pdb` 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 most popular pairwise sequence comparison algorithm for database searching, performs gapped local alignments via 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 the query and database lengths, and thus is comparable between BLASTP searches. Examining plots of BLAST alignment lengths, scores, E-values and normalized scores ( $-\log(\text{E-Value})$ ) from the `blast.pdb` function can aid in the identification sensible hit similarity thresholds. This is facilitated by the `plot.blast` function.

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 used 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

The function `blast.pdb` returns a list with three components, `hit.tbl`, `raw`, and `url`. The function `plot.blast` produces a plot on the active graphics device and returns a list object with four components, `hits`, `pdb.id`, `acc`, and `inds`. See below:

hit.tbl	<p>a data frame summarizing BLAST results for each reported hit. It contains following major columns:</p> <ul style="list-style-type: none"><li>• ‘bitscore’, a numeric vector containing the raw score for each alignment.</li><li>• ‘evalue’, a numeric vector containing the E-value of the raw score for each alignment.</li><li>• ‘mlog.evalue’, a numeric vector containing minus the natural log of the E-value.</li><li>• ‘acc’, a character vector containing the accession database identifier of each hit.</li><li>• ‘pdb.id’, a character vector containing the PDB database identifier of each hit.</li></ul>
raw	<p>a data frame containing the raw BLAST output. Note multiple hits may appear in the same row.</p>
url	<p>a single element character vector with the NCBI result URL and RID code. This can be passed to the get.blast function.</p>
hits	<p>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.</p>
pdb.id	<p>a character vector containing the PDB database identifier of each hit above the chosen threshold.</p>
acc	<p>a character vector containing the accession database identifier of each hit above the chosen threshold.</p>
inds	<p>a numeric vector containing the indices of the hits relative to the input blast object.</p>

### 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:

<https://www.ncbi.nlm.nih.gov/BLAST/>.

### See Also

[plot.blast](#), [hmmmer](#), [seqaln](#), [get.pdb](#)

### Examples

```
## Not run:
pdb <- read.pdb("4q21")
blast <- blast.pdb( pdbseq(pdb) )

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

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

# Examine and download 'best' hits
top.hits <- plot.blast(blast, cutoff=188)
head(top.hits$hits)
#get.pdb(top.hits)

## End(Not run)
```

---

bounds

*Bounds of a Numeric Vector*

---

### Description

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

### Usage

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

### Arguments

nums	a numeric vector.
dup.ind	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.ind' 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)
```

---

bounds.sse

*Obtain A SSE Object From An SSE Sequence Vector*

---

**Description**

Inverse process of the function [pdb2sse](#).

**Usage**

```
bounds.sse(x, pdb = NULL)
```

**Arguments**

`x` a character vector indicating SSE for each amino acid residue.  
`pdb` an object of class `pdb` as obtained from function [read.pdb](#). Can be ignored if `x` has 'names' attribute for residue labels.

**Details**

call for its effects.

**Value**

a 'sse' object.

**Note**

In both `$helix` and `$sheet`, an additional `$id` component is added to indicate the original numbering of the sse. This is particularly useful in e.g. `trim.pdb()` function.

**Author(s)**

Xin-Qiu Yao & Barry Grant

**See Also**

[pdb2sse](#)

**Examples**

```
# PDB server connection required - testing excluded

pdb <- read.pdb("1a7l")
sse <- pdb2sse(pdb)
sse.ind <- bounds.sse(sse)
sse.ind
```

---

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 ( $\geq 1$ ).

**Details**

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

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

**Value**

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

**Author(s)**

Barry Grant

**References**

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

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

**See Also**

[vmd\\_colors](#), [cm.colors](#), [colors](#), [palette](#), [hsv](#), [rgb](#), [gray](#), [col2rgb](#)

**Examples**

```
# Redundant testing excluded

# Color a distance matrix
pdb <- read.pdb( system.file("examples/1hel.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")
```

---

cat.pdb

*Concatenate Multiple PDB Objects*

---

**Description**

Produce a new concatenated PDB object from two or more smaller PDB objects.

**Usage**

```
cat.pdb(..., renumber=FALSE, rechain=TRUE)
```

**Arguments**

...	two or more PDB structure objects obtained from <a href="#">read.pdb</a> .
renumber	logical, if 'TRUE' residues will be renumbered.
rechain	logical, if 'TRUE' molecules will be assigned new chain identifiers.

**Details**

This is a basic utility function for creating a concatenated PDB object based on multiple smaller PDB objects.

**Value**

Returns an object of class "pdb". See [read.pdb](#) for further details.

**Author(s)**

Lars Skjaerven

**References**

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

**See Also**

[read.pdb](#), [atom.select](#), [write.pdb](#), [trim.pdb](#)

**Examples**

```
## Not run:
## Read a PDB file from the RCSB online database
pdb1 <- read.pdb("1et1")
pdb2 <- read.pdb("1hel")

## Concat
new.pdb <- cat.pdb(pdb1, pdb2, pdb1, rechain=TRUE, renumber=TRUE)

## Write to file
write.pdb(new.pdb, file="concat.pdb")

## End(Not run)
```

---

chain.pdb

*Find Possible PDB Chain Breaks*

---

**Description**

Find possible chain breaks based on connective Calpha or peptide bond (C-N) atom separation.

**Usage**

```
chain.pdb(pdb, ca.dist = 4, bond=TRUE, bond.dist=1.5, blank = "X", rtn.vec = TRUE)
```



**Arguments**

pdb	a PDB structure object obtained from <a href="#">read.pdb</a> .
ca.dist	the maximum distance that separates Calpha atoms considered to be in the same chain.
bond	logical, if TRUE inspect peptide bond (C-N) instead of Calpha-Calpha distances whenever possible.
bond.dist	cutoff value for C-N distance separation.
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 or peptide bond (C-N) atoms separated by at least `bond.dist`.

**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**

```
# PDB server connection required - testing excluded

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

---

check.utility	<i>Check on Missing Utility Programs</i>
---------------	--

---

**Description**

Internally used in examples, tests, or vignettes.

**Usage**

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

**Arguments**

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

**Details**

Check if requested utility programs are available or not.

**Value**

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

**Examples**

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

---

clean.pdb	<i>Inspect And Clean Up A PDB Object</i>
-----------	--

---

**Description**

Inspect alternative coordinates, chain breaks, bad residue numbering, non-standard/unknown amino acids, etc. Return a 'clean' pdb object with fixed residue numbering and optionally relabeled chain IDs, corrected amino acid names, removed water, ligand, or hydrogen atoms. All changes are recorded in a log in the returned object.

**Usage**

```
clean.pdb(pdb, consecutive = TRUE, force.renumber = FALSE,
         fix.chain = FALSE, fix.aa = FALSE, rm.wat = FALSE, rm.lig = FALSE,
         rm.h = FALSE, verbose = FALSE)
```

**Arguments**

pdb	an object of class pdb as obtained from function <a href="#">read.pdb</a> .
consecutive	logical, if TRUE renumbering will result in consecutive residue numbers spanning all chains. Otherwise new residue numbers will begin at 1 for each chain.
force.renumber	logical, if TRUE atom and residue records are renumbered even if no 'insert' code is found in the pdb object.
fix.chain	logical, if TRUE chains are relabeled based on chain breaks detected.
fix.aa	logical, if TRUE non-standard amino acid names are converted into equivalent standard names.
rm.wat	logical, if TRUE water atoms are removed.
rm.lig	logical, if TRUE ligand atoms are removed.
rm.h	logical, if TRUE hydrogen atoms are removed.
verbose	logical, if TRUE details of the conversion process are printed.

**Details**

call for its effects.

**Value**

a 'pdb' object with an additional \$log component storing all the processing messages.

**Author(s)**

Xin-Qiu Yao & Barry Grant

**See Also**

[read.pdb](#)

**Examples**

```
# PDB server connection required - testing excluded

pdb <- read.pdb("1a71")
clean.pdb(pdb)
```

cmap

*Contact Map***Description**

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

**Usage**

```
cmap(...)
```

## Default S3 method:  
cmap(...)

## S3 method for class 'xyz'  
cmap(xyz, grpby = NULL, dcut = 4, scut = 3, pcut=1, binary=TRUE,  
      mask.lower = TRUE, collapse=TRUE, gc.first=FALSE, ncore=1, nseg.scale=1, ...)

## S3 method for class 'pdb'  
cmap(pdb, inds = NULL, verbose = FALSE, ...)

## S3 method for class 'pdb'  
cmap(pdb, rm.gaps=FALSE, all.atom=FALSE, ...)

**Arguments**

xyz	numeric vector of xyz coordinates or a numeric matrix of coordinates with a row per structure/frame.
grpby	a vector counting connective duplicated elements that indicate the elements of xyz that should be considered as a group (e.g. atoms from a particular residue).
dcut	a cutoff distance value below which atoms are considered in contact.
scut	a cutoff neighbour value which has the effect of excluding atoms that are sequentially within this value.
pcut	a cutoff probability of structures/frames showing a contact, above which atoms are considered in contact with respect to the ensemble. Ignored if binary=FALSE.
binary	logical, if FALSE the raw matrix containing fraction of frames that two residues are in contact is returned.
mask.lower	logical, if TRUE the lower matrix elements (i.e. those below the diagonal) are returned as NA.
collapse	logical, if FALSE an array of contact maps for all frames is returned.
gc.first	logical, if TRUE will call gc() first before calculation of distance matrix. This is to solve the memory overload problem when ncore > 1 and xyz has many rows, with a bit sacrifice on speed.
ncore	number of CPU cores used to do the calculation. ncore>1 requires package 'parallel' installed.

nseg.scale	split input data into specified number of segments prior to running multiple core calculation. See <a href="#">fit.xyz</a> .
pdb	a structure object of class "pdb", obtained from <a href="#">read.pdb</a> .
inds	a list object of ATOM and XYZ indices as obtained from <a href="#">atom.select</a> .
verbose	logical, if TRUE details of the selection are printed.
pdbs	a 'pdbs' object as returned by <a href="#">read.fasta.pdb</a> , <a href="#">read.all</a> , or <a href="#">pdbaln</a> .
rm.gaps	logical, if TRUE gapped positions are removed in the returned value.
all.atom	logical, if TRUE all-atom coordinates from <a href="#">read.all</a> are used.
...	arguments passed to and from functions.

### Details

A contact map is a simplified distance matrix. See the distance matrix function [dm](#) for further details. Function "cmap.pdb" is a wrapper for "cmap.xyz" which selects all 'notwater' atoms and calculates the contact matrix grouped by residue number.

### Value

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

### Author(s)

Barry Grant

### References

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

### See Also

[dm](#), [dccm](#), [dist](#), [dist.xyz](#)

### Examples

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

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

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

## Not run:
##- Read Traj file
```

```

trj <- read.dcd( system.file("examples/hivp.dcd", package="bio3d") )
## For each frame of trajectory
sum.cont <- NULL
for(i in 1:nrow(trj)) {

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

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

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

## End(Not run)

```

cna

*Protein Dynamic Correlation Network Construction and Community Analysis.*

## Description

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

## Usage

```

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

```

## Arguments

cij	A numeric array with 2 dimensions (nXn) containing atomic correlation values, where "n" is the residue number. The matrix elements should be in between 0 and 1 (atomic correlations). Can be also a set of correlation matrices for ensemble network analysis. See ‘dcm’ function in bio3d package for further details.
...	Additional arguments passed to the methods cna.dcm and cna.ensmb.
cutoff.cij	Numeric element specifying the cutoff on cij matrix values. Coupling below cutoff.cij are set to 0.

<code>cm</code>	(optional) A numeric array with 2 dimensions (nXn) containing binary contact values, where "n" is the residue number. The matrix elements should be 1 if two residues are in contact and 0 if not in contact. See the 'cmap' function in bio3d package for further details.
<code>vnames</code>	A vector of names for each column in the input <code>cij</code> . This will be used for referencing residues in a similar way to residue numbers in later analysis.
<code>cluster.method</code>	A character string specifying the method for community determination. Supported methods are: <code>btwn="Girvan-Newman betweenness"</code> <code>walk="Random walk"</code> <code>greed="Greedy algorithm for modularity optimization"</code> <code>infomap="Infomap algorithm for community detection"</code>
<code>collapse.method</code>	A single element character vector specifying the 'cij' collapse method, can be one of 'max', 'median', 'mean', or 'trimmed'. By default the 'max' method is used to collapse the input residue based 'cij' matrix into a smaller community based network by taking the maximum 'abs(cij)' value between communities as the community-to-community <code>cij</code> value for clustered network construction.
<code>cols</code>	A vector of colors assigned to network nodes.
<code>minus.log</code>	Logical, indicating whether '-log(abs(cij))' values should be used for network construction.
<code>ncore</code>	Number of CPU cores used to do the calculation. By default, use all available cores.

## Details

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

Internally this function calls the igraph package functions 'graph.adjacency', 'edge.betweenness.community', 'walktrap.community', 'fastgreedy.community', and 'infomap.community'. The first constructs an undirected weighted network graph. The second performs Girvan-Newman style clustering by calculating the edge betweenness of the graph, removing the edge with the highest edge betweenness score, calculates modularity (i.e. the difference between the current graph partition and the partition of a random graph, see Newman and Girvan, Physical Review E (2004), Vol 69, 026113), then recalculating edge betweenness of the edges and again removing the one with the highest score, etc. The returned community partition is the one with the highest overall modularity value. 'walktrap.community' implements the Pons and Latapy algorithm based on the idea that random walks on a graph tend to get "trapped" into densely connected parts of it, i.e. a community. The random walk process is used to determine a distance between nodes. Nodes with low distance values are joined in the same community. 'fastgreedy.community' instead determines the community structure based on the optimization of the modularity. In the starting state each node is isolated and belongs to a separated community. Communities are then joined together (according to the network edges) in pairs and the modularity is calculated. At each step the join resulting in the highest increase of modularity is chosen. This process is repeated until a single community is obtained, then the

partitioning with the highest modularity score is selected. 'infomap.community' finds community structure that minimizes the expected description length of a random walker trajectory.

### Value

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

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

If an ensemble of correlation matrices is provided, a list of 'cna' object, of the 'ecna' class, will be returned.

### Author(s)

Guido Scarabelli and Barry Grant

### See Also

[plot.cna](#), [summary.cna](#), [vmd.cna](#), [graph.adjacency](#), [edge.betweenness.community](#), [walktrap.community](#), [fastgreedy.community](#), [infomap.community](#)

### Examples

```
# PDB server connection required - testing excluded

if (!requireNamespace("igraph", quietly = TRUE)) {
  message('Need igraph installed to run this example')
} else {

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

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

## Calculate correlations
cij <- dccm(modes)
#plot(cij, sse=pdb)
```



```

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

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

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

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

## select residues 24 to 27 and 85 to 90 in both chains
inds <- atom.select(pdb, resno=c(24:27,85:90), elety='CA')

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

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

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

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

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

}

```

**Description**

Find k shortest paths between a pair of nodes, source and sink, in a correlation network.

**Usage**

```

cnapath(cna, from, to=NULL, k=10, collapse=TRUE, ncore=NULL, ...)
## S3 method for class 'cnapath'
summary(object, ..., pdb = NULL, label = NULL, col = NULL,
  plot = FALSE, concise = FALSE, cutoff = 0.1, normalize = TRUE, weight = FALSE)
## S3 method for class 'cnapath'
print(x, ...)
## S3 method for class 'cnapath'
plot(x, ...)
## S3 method for class 'ecnapath'
plot(x, ...)

```

**Arguments**

cna	A 'cna' object or a list of 'cna' objects obtained from <a href="#">cna</a> .
from	Integer vector or matrix indicating node id(s) of source. If is matrix and to is NULL, the first column represents source and the second sink.
to	Integer vector indicating node id(s) of sink. All combinations of from and to values will be used as source/sink pairs.
k	Integer, number of suboptimal paths to identify.
collapse	Logical, if TRUE results from all source/sink pairs are merged with a single 'cnapath' object returned.
ncore	Number of CPU cores used to do the calculation. By default (NULL), use all detected CPU cores.
object	A 'cnapath' class of object obtained from cnapath. Multiple 'object' input is allowed for comparing paths from different networks.
pdb	A 'pdb' class of object obtained from <a href="#">read.pdb</a> and is used as the reference for node residue ids (in <code>summary.cnapath</code> ) or for molecular visualization with VMD (in <code>vmd.cnapath</code> ).
label	Character, label for paths identified from different networks.
col	Colors for plotting statistical results for paths identified from different networks.
plot	Logical, if TRUE path length distribution and node degeneracy will be plotted.
concise	Logical, if TRUE only 'on path' residues will be displayed in the node degeneracy plot.
cutoff	Numeric, nodes with node degeneracy larger than cutoff are shown in the output.
normalize	Logical, if TRUE node degeneracy is divided by the total (weighted) number of paths.
weight	Logical, if TRUE each path is weighted by path length in calculating the node degeneracy.
x	A 'cnapath' class object, or a list of such objects, as obtained from function cnapath.
...	Additional arguments passed to <a href="#">igraph</a> function <a href="#">get.shortest.paths</a> (in the function cnapath), passed to <code>summary.cnapath</code> (in <code>print.cnapath</code> ), as additional paths for comparison (in <code>summary.cnapath</code> ).

**Value**

The function `cnapath` returns a (or a list of) ‘cnapath’ class of list containing following three components:

<code>path</code>	a list object containing all identified suboptimal paths. Each entry of the list is a sequence of node ids for the path.
<code>epath</code>	a list object containing all identified suboptimal paths. Each entry of the list is a sequence of edge ids for the path.
<code>dist</code>	a numeric vector of all path lengths.

The function `summary.cnapath` returns a matrix of (normalized) node degeneracy for ‘on path’ residues.

**Author(s)**

Xin-Qiu Yao

**References**

Yen, J.Y. (1971) *Management Science* **17**, 712–716.

**See Also**

[cna](#), [cna.dccm](#), [vmd.cna](#), [vmd.cnapath](#), [get.shortest.paths](#).

**Examples**

```
# Redundant testing excluded

if (!requireNamespace("igraph", quietly = TRUE)) {
  message('Need igraph installed to run this example')
} else {

  attach(transducin)
  inds = match(c("1TND_A", "1TAG_A"), pdbs$id)

  npdbs <- trim(pdbs, row.inds=inds)
  gaps.res <- gap.inspect(npdbs$ali)

  modes <- nma(npdbs)
  cij <- dccm(modes)
  net <- cna(cij, cutoff.cij=0.3)

  # get paths
  pa1 <- cnapath(net[[1]], from = 314, to=172, k=50)
  pa2 <- cnapath(net[[2]], from = 314, to=172, k=50)

  # print the information of a path
  pa1
```

```

# print two paths simultaneously
pas <- list(pa1, pa2)
names(pas) <- c("GTP", "GDP")
print.cnapath(pas)

# Or, for the same effect,
# summary(pa1, pa2, label=c("GTP", "GDP"))

# replace node numbers with residue name and residue number in the PDB file
pdb <- read.pdb("1tnd")
pdb <- trim.pdb(pdb, atom.select(pdb, chain="A", resno=npdbs$resno[1, gaps.res$f.inds]))
print.cnapath(pas, pdb=pdb)

# plot path length distribution and node degeneracy
print.cnapath(pas, pdb = pdb, col=c("red", "darkgreen"), plot=TRUE)

# View paths in 3D molecular graphic with VMD
#vmd.cnapath(pa1, pdb, launch = TRUE)
#vmd.cnapath(pa1, pdb, colors = 7, launch = TRUE)
#vmd.cnapath(pa1, pdb, spline=TRUE, colors=c("pink", "red"), launch = TRUE)
#pdb2 <- read.pdb("1tag")
#pdb2 <- trim.pdb(pdb2, atom.select(pdb2, chain="A", resno=npdbs$resno[2, gaps.res$f.inds]))
#vmd.cnapath(pa2, pdb2, launch = TRUE)

detach(transducin)

}

```

---

com

*Center of Mass*


---

## Description

Calculate the center of mass of a PDB object.

## Usage

```

com(...)

## S3 method for class 'pdb'
com(pdb, inds=NULL, use.mass=TRUE, ...)

## S3 method for class 'xyz'
com(xyz, mass=NULL, ...)

```

## Arguments

`pdb` an object of class `pdb` as obtained from function `read.pdb`.

inds	atom and xyz coordinate indices obtained from <code>atom.select</code> that selects the elements of <code>pdb</code> upon which the calculation should be based.
use.mass	logical, if TRUE the calculation will be mass weighted (center of mass).
...	additional arguments to <code>atom2mass</code> .
xyz	a numeric vector or matrix of Cartesian coordinates (e.g. an object of type <code>xyz</code> ).
mass	a numeric vector containing the masses of each atom in <code>xyz</code> .

### Details

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

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

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

### Value

Returns the Cartesian coordinates at the center of mass.

### Author(s)

Lars Skjaerven

### References

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

### See Also

[read.pdb](#), [atom2mass](#)

### Examples

```
# PDB server connection required - testing excluded

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

## Center of mass:
com(pdb)

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

```

## using XYZ Cartesian coordinates
xyz <- pdb$xyz[, inds$xyz]
com.xyz(xyz)

## with mass weighting
com.xyz(xyz, mass=atom2mass(pdb$atom[inds$atom, "eleyt"]) )

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

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

myeleyt <- data.frame(name = c("CL2", "PT1", "N1", "N2"), symb = c("Cl", "Pt", "N", "N"))
mymasses <- data.frame(symb = c("Cl", "Pt"), mass = c(35.45, 195.08))
mycom <- com(pdb, inds, eleyt.custom=myeleyt, mass.custom=mymasses)

## End(Not run)

```

---

combine.select

*Combine Atom Selections From PDB Structure*


---

## Description

Do "and", "or", or "not" set operations between two or more atom selections made by [atom.select](#)

## Usage

```
combine.select(sel1=NULL, sel2=NULL, ..., operator="AND", verbose=TRUE)
```

## Arguments

sel1	an atom selection object of class "select", obtained from <a href="#">atom.select</a> .
sel2	a second atom selection object of class "select", obtained from <a href="#">atom.select</a> .
...	more select objects for the set operation.
operator	name of the set operation.
verbose	logical, if TRUE details of the selection combination are printed.

## Details

The value of operator should be one of following: (1) "AND", "and", or "&" for set intersect, (2) "OR", "or", "I", or "+" for set union, (3) "NOT", "not", "!", or "-" for set difference  $se11 - se12 - se13 \dots$

## Value

Returns a list of class "select" with components:

atom	atom indices of selected atoms.
xyz	xyz indices of selected atoms.
call	the matched call.

## Author(s)

Xin-Qiu Yao

## References

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

## See Also

[atom.select](#), [as.select](#) [read.pdb](#), [trim.pdb](#)

## Examples

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

## - Build atom selections to be operated
# Select C-alpha atoms of entire system
ca.global.inds <- atom.select(pdb, "calpha")

# Select C-beta atoms of entire protein
cb.global.inds <- atom.select(pdb, "protein", eley="CB")

# Select backbone atoms of entire system
bb.global.inds <- atom.select(pdb, "backbone")

# Select all atoms with residue number from 46 to 50
aa.local.inds <- atom.select(pdb, resno=46:50)

# Do set intersect:
# - Return C-alpha atoms with residue number from 46 to 50
ca.local.inds <- combine.select(ca.global.inds, aa.local.inds)
print( pdb$atom[ ca.local.inds$atom, ] )

# Do set subtract:
```

```

# - Return side-chain atoms with residue number from 46 to 50
sc.local.inds <- combine.select(aa.local.inds, bb.global.inds, operator="-")
print( pdb$atom[ sc.local.inds$atom, ] )

# Do set union:
# - Return C-alpha and side-chain atoms with residue number from 46 to 50
casc.local.inds <- combine.select(ca.local.inds, sc.local.inds, operator="+")
print( pdb$atom[ casc.local.inds$atom, ] )

# More than two selections:
# - Return side-chain atoms (but not C-beta) with residue number from 46 to 50
sc2.local.inds <- combine.select(aa.local.inds, bb.global.inds, cb.global.inds, operator="-")
print( pdb$atom[ sc2.local.inds$atom, ] )

```

---

community.aln

Align communities from two or more networks

---

## Description

Find equivalent communities from two or more networks and re-assign colors to them in a consistent way across networks. A ‘new.membership’ vector is also generated for each network, which maps nodes to community IDs that are renumbered according to the community equivalency.

## Usage

```
community.aln(x, ..., aln = NULL)
```

## Arguments

x, ...	two or more objects of class <code>cna</code> (if the numbers of nodes are different, an alignment ‘fasta’ object is required for the <code>aln</code> argument; See below) as obtained from function <a href="#">cna</a> . Alternatively, a list of <code>cna</code> objects can be given to x.
aln	alignment for comparing networks with different numbers of nodes.

## Details

This function facilitates the inspection on the variance of the community partition in a group of similar networks. The original community numbering (and so the colors of communities in the output of `plot.cna` and `vmd.cna`) can be inconsistent across networks, i.e. equivalent communities may display different colors, impeding network comparison. The function calculates the dissimilarity between all communities and clusters communities with ‘hclust’ function. In each cluster, 0 or 1 community per network is included. The color attribute of communities is then re-assigned according to the clusters through all networks. In addition, a ‘new.membership’ vector is generated for each network, which maps nodes to new community IDs that are numbered consistently across networks.



**Value**

Returns a list of updated cna objects.

**See Also**

[cna](#), [plot.cna](#), [vmd.cna](#)

**Examples**

```
# Needs MUSCLE installed - testing excluded
if(check.utility("muscle")) {

  if (!requireNamespace("igraph", quietly = TRUE)) {
    message('Need igraph installed to run this example')
  } else {

    ## Fetch PDB files and split to chain A only PDB files
    ids <- c("1tnd_A", "1tag_A")
    files <- get.pdb(ids, split = TRUE, path = tempdir())

    ## Sequence Alignment
    pdbs <- pdbaln(files, outfile = tempfile())

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

    ## Dynamic Cross Correlation Matrix
    cijs <- dccm(modes)$all.dccm

    ## Correlation Network
    nets <- cna(cijs, cutoff.cij=0.3)

    ## Align network communities
    nets.aln <- community.aln(nets)

    ## plot all-residue and coarse-grained (community) networks
    pdb <- pdbs2pdb(pdbs, inds=1, rm.gaps=TRUE)[[1]]
    op <- par(no.readonly=TRUE)

    # before alignment
    par(mar=c(0.1, 0.1, 0.1, 0.1), mfrow=c(2,2))
    invisible( lapply(nets, function(x)
      plot(x, layout=layout.cna(x, pdb=pdb, k=3, full=TRUE)[, 1:2],
        full=TRUE)) )
    invisible( lapply(nets, function(x)
      plot(x, layout=layout.cna(x, pdb=pdb, k=3)[, 1:2])) )

    # after alignment
    par(mar=c(0.1, 0.1, 0.1, 0.1), mfrow=c(2,2))
    invisible( lapply(nets.aln, function(x)
      plot(x, layout=layout.cna(x, pdb=pdb, k=3, full=TRUE)[, 1:2],
```

```

        full=TRUE)) )
invisible( lapply(nets.aln, function(x)
  plot(x, layout=layout.cna(x, pdb=pdb, k=3)[, 1:2])) )

par(op)

}
}

```

---

community.tree	<i>Reconstruction of the Girvan-Newman Community Tree for a CNA Class Object.</i>
----------------	---

---

## Description

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

## Usage

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

## Arguments

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

## Details

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

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

## Value

Returns a list object that includes the following components:

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

**Author(s)**

Guido Scarabelli

**See Also**[cna](#), [network.amendment](#), [summary.cna](#)**Examples**

```

# PDB server connection required - testing excluded

if (!requireNamespace("igraph", quietly = TRUE)) {
  message('Need igraph installed to run this example')
} else {

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

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

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

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

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

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

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

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

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

}

```

---

consensus

*Sequence Consensus for an Alignment*

---

### Description

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

### Usage

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

### Arguments

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

### Value

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

### Author(s)

Barry Grant

### References

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

### See Also

[read.fasta](#)

### Examples

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

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

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

```

aa <- rev(rownames(con$freq))

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

# Add consensus along the axis
axis(side=1, at=seq(0,length(con$seq),by=5))
axis(side=2, at=c(1:22), labels=aa)
axis(side=3, at=c(1:length(con$seq)), labels =con$seq)
axis(side=4, at=c(1:22), labels=aa)
grid(length(con$seq), length(aa))
box()

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

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

```

---

 conserv

*Score Residue Conservation At Each Position in an Alignment*


---

### Description

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

### Usage

```

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

```

### Arguments

x	an alignment list object with id and ali components, similar to that generated by <a href="#">read.fasta</a> .
method	the conservation assesment method.
sub.matrix	a matrix to score conservation.
matrix.file	a file name of an arbitrary user matrix.
normalize.matrix	logical, if TRUE the matrix is normalized prior to assessing conservation.

## Details

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

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

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

“Entropy” is based on Shannons information entropy. See the [entropy](#) function for further details.

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

## Value

Returns a numeric vector of scores

## Note

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

## Author(s)

Barry Grant

## References

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

## See Also

[read.fasta](#), [read.fasta.pdb](#)

## Examples

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

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

---

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

---

### Description

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

### Usage

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

### Arguments

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

### Details

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

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

### Value

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

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

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

**Note**

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

**Author(s)**

Barry Grant

**References**

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

For a description of PDB format (version3.3) see:

<http://www.wwpdb.org/documentation/format33/v3.3.html>.

**See Also**

[atom.select](#), [write.pdb](#), [read.dcd](#), [read.fasta.pdb](#), [read.fasta](#)

**Examples**

```
## Not run:

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

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

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

## End(Not run)
```



---

`core.cmap`*Identification of Contact Map Core Positions*

---

**Description**

Find core positions that have the largest number of contact with neighboring residues.

**Usage**

```
core.cmap(pdb, write.pdb = FALSE, outfile="core.pdb",  
          cutoff = NULL, refine = FALSE, ncore = NULL, ...)
```

**Arguments**

<code>pdbs</code>	an alignment data structure of class 'pdbs' as obtained with <a href="#">read.fasta.pdb</a> or <a href="#">pdbaln</a> , or a numeric matrix of aligned C-alpha xyz Cartesian coordinates.
<code>write.pdb</code>	logical, if TRUE core coordinate files, containing only core positions for each iteration, are written to a location specified by <code>outfile</code> .
<code>outfile</code>	character string specifying the output directory when <code>write.pdb</code> is 'TRUE'.
<code>cutoff</code>	numeric value specifying the inclusion criteria for core positions.
<code>refine</code>	logical, if TRUE explore core positions determined by multiple eigenvectors. By default only the eigenvector describing the largest variation is used.
<code>ncore</code>	number of CPU cores used to do the calculation. By default ( <code>ncore=NULL</code> ) use all cores detected.
<code>...</code>	arguments passed to and from functions.

**Details**

This function calculates eigenvector centrality of the weighted contact network built based on input structure data and uses it to determine the core positions.

In this context, core positions correspond to the most invariant C-alpha atom positions across an aligned set of protein structures. Traditionally one would use the `core.find` function to for their identification and then use these positions as the basis for improved structural superposition. This more recent function utilizes a much faster approach and is thus preferred in time sensitive applications such as shiny apps.

**Value**

Returns a list of class "select" containing 'atom' and 'xyz' indices.

**Author(s)**

Xin-Qiu Yao

## References

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

## See Also

[core.find](#), [read.fasta.pdb](#), [fit.xyz](#)

## Examples

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

##-- Find 'core' positions
core <- core.cmap(pdbs)
xyz <- pdbfit(pdbs, core, outputPath="corefit_structures")

## End(Not run)
```

---

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(...)

## S3 method for class 'pdbs'
core.find(pdbs, shortcut = FALSE, rm.island = FALSE,
          verbose = TRUE, stop.at = 15, stop.vol = 0.5,
          write.pdbs = FALSE, outputPath="core_pruned",
          ncore = 1, nseg.scale = 1, progress = NULL, ...)

## Default S3 method:
core.find(xyz, ...)

## S3 method for class 'pdb'
core.find(pdb, verbose=TRUE, ...)
```

**Arguments**

pdbs	a numeric matrix of aligned C-alpha xyz Cartesian coordinates. For example an alignment data structure obtained with <a href="#">read.fasta.pdb</a> or <a href="#">pdbaln</a> .
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 iteration 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 <code>write.pdbs</code> is TRUE.
ncore	number of CPU cores used to do the calculation. <code>ncore&gt;1</code> requires package ‘parallel’ installed.
nseg.scale	split input data into specified number of segments prior to running multiple core calculation. See <a href="#">fit.xyz</a> .
progress	progress bar for use with shiny web app.
xyz	a numeric matrix of xyz Cartesian coordinates, e.g. obtained from <a href="#">read.dcd</a> or <a href="#">read.ncdf</a> .
pdb	an object of type <code>pdb</code> as obtained from function <a href="#">read.pdb</a> with multiple frames ( $\geq 4$ ) stored in its <code>xyz</code> component. Note that the function will attempt to identify C-alpha and phosphate atoms (for protein and nucleic acids, respectively) in which the calculation should be based.
...	arguments passed to and from functions.

**Details**

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

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

**Value**

Returns a list of class “core” with the following components:

volume	total core volume at each fitting iteration/round.
length	core length at each round.

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

**Note**

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

**Author(s)**

Barry Grant

**References**

- Grant, B.J. et al. (2006) *Bioinformatics* **22**, 2695–2696.  
 Gerstein and Altman (1995) *J. Mol. Biol.* **251**, 161–175.  
 Gerstein and Chothia (1991) *J. Mol. Biol.* **220**, 133–149.

**See Also**

[read.fasta.pdb](#), [plot.core](#), [fit.xyz](#)

**Examples**

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

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

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

```

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

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

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

## End(Not run)

## Not run:
##-- Run core.find() on a multimodel PDB file
pdb <- read.pdb('1d1d', multi=TRUE)
core <- core.find(pdb)

##-- Run core.find() on a trajectory
trtfile <- system.file("examples/hivp.dcd", package="bio3d")
trj <- read.dcd(trtfile)

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

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

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

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

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

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

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

## End(Not run)

```

---

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

---

**Description**

Calculate the covariance matrix from a normal mode object.

**Usage**

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

**Arguments**

<code>nma</code>	an <code>nma</code> object as obtained from function <code>nma.pdb</code> .
<code>enma</code>	an <code>enma</code> object as obtained from function <code>nma.pdbs</code> .
<code>ncore</code>	number of CPU cores used to do the calculation. <code>ncore&gt;1</code> requires package ‘parallel’ installed.

**Details**

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

**Value**

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

**Author(s)**

Lars Skjaerven

**References**

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

**See Also**

[nma](#)

---

covoverlap	<i>Covariance Overlap</i>
------------	---------------------------

---

**Description**

Calculate the covariance overlap obtained from NMA.

**Usage**

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

**Arguments**

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

**Details**

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

**Value**

Returns the similarity coefficient(s).

**Author(s)**

Lars Skjaerven

## References

Skjaerven, L. et al. (2014) *BMC Bioinformatics* **15**, 399. Grant, B.J. et al. (2006) *Bioinformatics* **22**, 2695–2696. Romo, T.D. et al. (2011) *Proteins* **79**, 23–34.

## See Also

Other similarity measures: [sip](#), [covoverlap](#), [bhattacharyya](#).

---

dccb

*DCCM: Dynamical Cross-Correlation Matrix*

---

## Description

Determine the cross-correlations of atomic displacements.

## Usage

```
dccb(x, ...)
```

## Arguments

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

## Details

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

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

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

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

`plot.dccb` and `pymol.dccb` provides convenient functionality to plot a correlation map, and visualize the correlations in the structure, respectively.

See examples for each corresponding function for more details.

## Author(s)

Barry Grant, Lars Skjaerven



**References**

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

**See Also**

[dcm.xyz](#), [dcm.nma](#), [dcm.enma](#), [dcm.pca](#), [plot.dcm](#), [pymol.dcm](#).

---

dcm.enma

*Cross-Correlation for Ensemble NMA (eNMA)*

---

**Description**

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

**Usage**

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

**Arguments**

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

**Details**

This is a wrapper function for calling `dcm.nma` on a collection of ‘`nma`’ objects as obtained from function `nma.pdbs`.

See examples for more details.

**Value**

Returns a list with the following components:

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

**Author(s)**

Lars Skjaerven

## References

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

## See Also

[nma](#), [dccm.nma](#), [plot.dccm](#)

## Examples

```
## Needs MUSCLE installed - testing excluded

if(check.utility("muscle")) {

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

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

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

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

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

}
```

---

dccm.gnm

*Dynamic Cross-Correlation from Gaussian Network Model*

---

## Description

Calculate the cross-correlation matrix from Gaussian network model normal modes analysis.

## Usage

```
## S3 method for class 'gnm'
dccm(x, ...)

## S3 method for class 'egnm'
dccm(x, ...)
```

## Arguments

x                    an object of class 'gnm' or 'egnm' as obtained from [gnm](#).  
...                    additional arguments (currently ignored).

## Details

This function calculates the cross-correlation matrix from Gaussian network model (GNM) normal modes analysis (NMA) obtained from [gnm](#). It returns a matrix of residue-wise cross-correlations whose elements,  $C_{ij}$ , may be displayed in a graphical representation frequently termed a dynamical cross-correlation map, or DCCM. (See more details in `help(dcm.nma)`).

## Value

Returns a cross-correlation matrix.

## Author(s)

Xin-Qiu Yao & Lars Skjaerven

## References

Bahar, I. et al. (1997) *Folding Des.* **2**, 173.

## See Also

[gnm](#), [dcm.nma](#), [dcm.enma](#), [plot.dcm](#).

## Examples

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

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

## Calculate correlation matrix
cm <- dcm(modes)

## Plot correlation map
plot(cm, sse = pdb, contour = FALSE, col.regions = bwr.colors(20),
      at = seq(-1, 1, 0.1))
```

---

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

---

**Description**

Calculate the cross-correlation matrix from Normal Modes Analysis.

**Usage**

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

**Arguments**

<code>x</code>	an object of class <code>nma</code> as obtained from function <code>nma</code> .
<code>nmodes</code>	numerical, number of modes to consider.
<code>ncore</code>	number of CPU cores used to do the calculation. <code>ncore&gt;1</code> requires package ‘parallel’ installed.
<code>progress</code>	progress bar for use with shiny web app.
<code>...</code>	additional arguments ?

**Details**

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

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

**Value**

Returns a cross-correlation matrix.

**Author(s)**

Lars Skjaerven

**References**

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

**See Also**

[nma](#), [plot.dccm](#)

**Examples**

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

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

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

## Plot correlation map
plot(cm, sse = pdb, contour = FALSE, col.regions = bwr.colors(20),
      at = seq(-1, 1, 0.1))
```

---

dccb.pca	<i>Dynamical Cross-Correlation Matrix from Principal Component Analysis</i>
----------	---

---

**Description**

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

**Usage**

```
## S3 method for class 'pca'
dccb(x, pc = NULL, method = c("pearson", "lmi"), ncore = NULL, ...)
```

**Arguments**

x	an object of class <code>pca</code> as obtained from function <code>pca.xyz</code> .
pc	numerical, indices of PCs to be included in the calculation. If all negative, PCs complementary to <code>abs(pc)</code> are included.
method	method to calculate the cross-correlation. Currently supports Pearson and linear mutual information (LMI).
ncore	number of CPU cores used to do the calculation. By default ( <code>ncore = NULL</code> ), use all available cores detected.
...	Additional arguments to be passed (currently ignored).

**Details**

This function calculates the cross-correlation matrix from principal component analysis (PCA) obtained from `pca.xyz` of a set of protein structures. It is an alternative way to calculate correlation in addition to the conventional way from xyz coordinates directly. But, in this new way one can freely chooses the PCs to be included in the calculation (e.g. for filtering out PCs with small eigenvalues).

**Value**

Returns a cross-correlation matrix with values in a range from -1 to 1 (Pearson) or from 0 to 1 (LMI).

**Author(s)**

Xin-Qiu Yao

**References**

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

**See Also**

[pca.xyz](#), [plot.dccm](#), [dccm](#), [dccm.xyz](#), [dccm.nma](#), [dccm.enma](#).

**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, resno=c(24:27,85:90), elety='CA')

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

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

## DCCM: only use first 10 PCs
cij <- dccm(pca, pc = c(1:10))

## Plot DCCM
plot(cij)

## DCCM: remove first 10 PCs
cij <- dccm(pca, pc = -c(1:10))

## Plot DCCM
plot(cij)
```

**Description**

Determine the cross-correlations of atomic displacements.

**Usage**

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

**Arguments**

x	a numeric matrix of Cartesian coordinates with a row per structure/frame.
reference	The reference structure about which displacements are analysed.
grpby	a vector counting connective duplicated elements that indicate the elements of xyz that should be considered as a group (e.g. atoms from a particular residue).
method	method to calculate the cross-correlation. Currently supports Pearson and linear mutual information (LMI).
ncore	number of CPU cores used to do the calculation. ncore=NULL will use all the cores detected.
nseg.scale	split input data into specified number of segments prior to running multiple core calculation. See <a href="#">fit.xyz</a> .
...	Additional arguments to be passed (currently ignored).

**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,  $C_{ij}$ , may be displayed in a graphical representation frequently termed a dynamical cross-correlation map, or DCCM.

If  $C_{ij} = 1$  the fluctuations of atoms  $i$  and  $j$  are completely correlated (same period and same phase), if  $C_{ij} = -1$  the fluctuations of atoms  $i$  and  $j$  are completely anticorrelated (same period and opposite phase), and if  $C_{ij} = 0$  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  $C_{ij}$  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.

If method = "pearson", the conventional Pearson's inner-product correlation calculation will be invoked, in which only the diagonal of each atom-atom variance-covariance sub-matrix is considered.

If method = "lmi", then the linear mutual information cross-correlation will be calculated. 'LMI' considers both diagonal and off-diagonal entries in the sub-matrices, and so even captures the correlation of atoms moving in orthogonal directions.

### Value

Returns a cross-correlation matrix with values in a range from -1 to 1 (Pearson) or from 0 to 1 (LMI).

### Author(s)

Xin-Qiu Yao, Hongyang Li, Gisle Saelensminde, and Barry Grant

### References

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

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

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

### See Also

[cor](#) for examining xyz cross-correlations, [dccm](#), [dccm.nma](#), [dccm.pca](#), [dccm.enma](#).

### 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, resno=c(24:27,85:90), eley='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=FALSE, 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")

## LMI matrix
cij <- dccm(xyz, method='lmi')

## Plot LMI matrix
#plot(cij)
col.scale <- colorRampPalette(c("gray95", "cyan"))(5)
plot(cij, at=seq(0.4,1, length=5), col.regions=col.scale)

```

---

deformation.nma

*Deformation Analysis*


---

## Description

Calculate deformation energies from Normal Mode Analysis.

## Usage

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

## Arguments

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

## Details

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

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

See examples for more details.

**Value**

Returns a list with the following components:

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

**Author(s)**

Lars Skjaerven

**References**

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

**See Also**

[nma](#)

**Examples**

```
# Running the example takes some time - testing excluded

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

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

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

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

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

## End(Not run)
```

---

`diag.ind`*Diagonal Indices of a Matrix*

---

**Description**

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

**Usage**

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

**Arguments**

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

**Details**

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

**Value**

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

**Author(s)**

Barry Grant

**References**

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

**See Also**

[diag](#), [lower.tri](#), [upper.tri](#), [matrix](#)

**Examples**

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

---

difference.vector      *Difference Vector*

---

### Description

Define a difference vector between two conformational states.

### Usage

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

### Arguments

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

### Details

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

### Value

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

### Author(s)

Lars Skjaerven

### References

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

### See Also

[overlap](#)

### Examples

```
attach(kinesin)

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

#-- Do PCA
```

```

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

# Define a difference vector between two structural states
diff.inds <- c(grep("d1v8ka", pdb$id),
              grep("d1goja", pdb$id))

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

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

detach(kinesin)

```

---

dist.xyz

---

*Calculate the Distances Between the Rows of Two Matrices*


---

### Description

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

### Usage

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

### Arguments

a	a 'xyz' object, numeric data matrix, or vector.
b	an optional second 'xyz' object, 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 corresponding rows of 'a' and 'b' are computed.
ncore	number of CPU cores used to do the calculation. ncore>1 requires package 'parallel' installed.
nseg.scale	split input data into specified number of segments prior to running multiple core calculation. See <a href="#">fit.xyz</a> .

### 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(...)  
  
## S3 method for class 'pdb'  
dm(pdb, inds = NULL, grp = TRUE, verbose=TRUE, ...)  
## S3 method for class 'pdbs'  
dm(pdbs, rm.gaps=FALSE, all.atom=FALSE,  
    aligned.atoms.only=NULL, ...)  
  
## S3 method for class 'xyz'  
dm(xyz, grpby = NULL, scut = NULL, mask.lower = TRUE,  
    gc.first=FALSE, ncore=1, ...)
```

**Arguments**

<code>pdb</code>	a <code>pdb</code> structure object as returned by <code>read.pdb</code> or a numeric vector of 'xyz' coordinates.
<code>inds</code>	atom and xyz coordinate indices obtained from <code>atom.select</code> that selects the elements of <code>pdb</code> upon which the calculation should be based.
<code>grp</code>	logical, if TRUE atomic distances will be grouped according to their residue membership. See 'grpby'.
<code>verbose</code>	logical, if TRUE possible warnings are printed.
<code>pdbs</code>	a 'pdbs' object as returned by <code>read.fasta.pdb</code> , <code>read.all</code> , or <code>pdbsaln</code> .
<code>rm.gaps</code>	logical, if TRUE gapped positions are removed in the returned value.
<code>all.atom</code>	logical, if TRUE all-atom coordinates from <code>read.all</code> are used.
<code>aligned.atoms.only</code>	logical, if TRUE only equivalent (aligned) atoms are considered. Only meaningful when <code>all.atom=TRUE</code> . Default: FALSE.
<code>xyz</code>	a numeric vector or matrix of Cartesian coordinates.
<code>grpby</code>	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).
<code>scut</code>	a cutoff neighbour value which has the effect of excluding atoms, or groups, that are sequentially within this value.
<code>mask.lower</code>	logical, if TRUE the lower matrix elements (i.e. those below the diagonal) are returned as NA.
<code>gc.first</code>	logical, if TRUE will call <code>gc()</code> first before calculation of distance matrix. This is to solve the memory overload problem when <code>ncore &gt; 1</code> and <code>xyz</code> has many rows/columns, with a bit sacrifice on speed.
<code>ncore</code>	number of CPU cores used to do the calculation. <code>ncore &gt; 1</code> requires package 'parallel' installed.
<code>...</code>	arguments passed to and from functions.

**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. With multiple frames the output is provided in a three dimensional array.

**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/residue name/element number/element name/; see `atom.select` for details.

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

**Author(s)**

Barry Grant

**References**

- Grant, B.J. et al. (2006) *Bioinformatics* **22**, 2695–2696.  
 Phillips (1970) *Biochem. Soc. Symp.* **31**, 11–28.  
 Holm (1993) *J. Mol. Biol.* **233**, 123–138.  
 Havel (1983) *Bull. Math. Biol.* **45**, 665–720.

**See Also**

`plot.dmat`, `read.pdb`, `atom.select`

**Examples**

```
# PDB server connection required - testing excluded

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

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

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

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

# Calculate DDM
c <- a - b

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

plot(c, axis.tick.space=10,
```



```

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

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

## End(Not run)

```

---

dssp

*Secondary Structure Analysis with DSSP or STRIDE*


---

## Description

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

## Usage

```

dssp(...)

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

## S3 method for class 'pdbs'
dssp(pdbs, ...)

## S3 method for class 'xyz'
dssp(xyz, pdb, ...)

stride(pdb, exefile = "stride", resno=TRUE)

## S3 method for class 'sse'
print(x, ...)

```

## Arguments

pdb	a structure object of class "pdb", obtained from <a href="#">read.pdb</a> .
exefile	file path to the 'DSSP' or 'STRIDE' program on your system (i.e. how is 'DSSP' or 'STRIDE' invoked).
resno	logical, if TRUE output is in terms of residue numbers rather than residue index (position in sequence).

full	logical, if TRUE bridge pairs and hbonds columns are parsed.
verbose	logical, if TRUE 'DSSP' warning and error messages are printed.
pdbs	a list object of class "pdbs" (obtained with <code>pdbaln</code> or <code>read.fasta.pdb</code> ).
xyz	a trajectory object of class "xyz", obtained from <code>read.ncdf</code> , <code>read.dcd</code> , <code>read.crd</code> .
x	an sse object obtained from <code>dssp.pdb</code> or <code>stride</code> .
...	additional arguments to and from functions.

### Details

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

### Value

Returns a list with the following components:

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

### Note

A system call is made to the 'DSSP' or 'STRIDE' program, which must be installed on your system and in the search path for executables. See [http://thegrantlab.org/bio3d/articles/online/install\\_vignette/Bio3D\\_install.html](http://thegrantlab.org/bio3d/articles/online/install_vignette/Bio3D_install.html) for instructions of how to install these programs.

For the hbonds list component the column names can be used as a convenient means of data access, namely:

Bridge pair 1 "BP1",  
 Bridge pair 2 "BP2",  
 Backbone H-bond (NH→O) "NH-O.1",  
 H-bond energy of NH→O "E1",  
 Backbone H-bond (O→NH) "O-HN.1",  
 H-bond energy of O→NH "E2",  
 Backbone H-bond (NH→O) "NH-O.2",  
 H-bond energy of NH→O "E3",

Backbone H-bond (O->NH) "O-HN.2",  
H-bond energy of O->NH "E4".

If 'resno=TRUE' the following additional columns are included:

Chain ID of resno "BP1": "ChainBP1",  
Chain ID of resno "BP2": "ChainBP2",  
Chain ID of resno "O-HN.1": "Chain1",  
Chain ID of resno "NH-O.2": "Chain2",  
Chain ID of resno "O-HN.1": "Chain3",  
Chain ID of resno "NH-O.2": "Chain4".

### Author(s)

Barry Grant, Lars Skjaerven (dssp.pdbs)

### 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:

<https://swift.cmbi.umcn.nl/gv/dssp/>.

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

For information on obtaining the 'STRIDE' program, see:

<http://webclu.bio.wzw.tum.de/stride/>, or copy it from an installation of VMD.

### See Also

[read.pdb](#), [torsion.pdb](#), [torsion.xyz](#), [plot.bio3d](#),  
[read.ncdf](#), [read.dcd](#), [read.prmtop](#), [read.crd](#),

### Examples

```
## Not run:
##- PDB example
# Read a PDB file
pdb <- read.pdb("1bg2")
sse <- dssp(pdb)
sse2 <- stride(pdb)

## Short summary
sse
sse2

# Helix data
sse$helix

# Percent SSE content
sum(sse$helix$length)/sum(pdb$alpha) * 100
sum(sse$sheet$length)/sum(pdb$alpha) * 100
```

```

##- PDBs example
aln <- read.fasta( system.file("examples/kif1a.fa",package="bio3d") )
pdbs <- read.fasta.pdb( aln )

## Aligned PDB defined secondary structure
pdbs$sse

## Aligned DSSP defined secondary structure
sse <- dssp(pdbs)

##- XYZ Trajectory
pdb <- read.pdb("2mda", multi=TRUE)
dssp.xyz(pdb$xyz, pdb)

## Note. for large MD trajectories you may want to skip some frames, e.g.
xyz <- rbind(pdb$xyz, pdb$xyz)      ## dummy trajectory
frames <- seq(1, to=nrow(xyz), by=4) ## frame numbers to examine
ss <- dssp.xyz(xyz[frames, ], pdb)  ## matrix of sse frame x residue

## End(Not run)

```

---

elements

*Periodic Table of the Elements*


---

## Description

This data set gives various information on chemical elements.

## Usage

```
elements
```

## Format

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

```

num atomic number
symb elemental symbol
areneg Allred and Rochow electronegativity (0.0 if unknown)
rcov covalent radii (in Angstrom) (1.6 if unknown)
rbo "bond order" radii
rvdw van der Waals radii (in Angstrom) (2.0 if unknown)
maxbnd maximum bond valence (6 if unknown)

```

mass IUPAC recommended atomic masses (in amu)  
 elneg Pauling electronegativity (0.0 if unknown)  
 ionization ionization potential (in eV) (0.0 if unknown)  
 elaffinity electron affinity (in eV) (0.0 if unknown)  
 red red value for visualization  
 green green value for visualization  
 blue blue value for visualization  
 name element name

### Source

Open Babel (2.3.1) file: element.txt

Created from the Blue Obelisk Cheminformatics Data Repository

Direct Source: <http://www.blueobelisk.org/>

<http://www.blueobelisk.org/repos/blueobelisk/elements.xml> includes further bibliographic citation information

- Allred and Rochow Electronegativity from <http://www.hull.ac.uk/chemistry/electroneg.php?type=Allred-Rochow>

- Covalent radii from <http://dx.doi.org/10.1039/b801115j>

- Van der Waals radii from <http://dx.doi.org/10.1021/jp8111556>

### Examples

```

data(elements)
elements

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

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

```

---

entropy

*Shannon Entropy Score*

---

### Description

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

**Usage**

```
entropy(alignment)
```

**Arguments**

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

**Details**

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

**Value**

Returns a list with five components:

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

**Note**

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

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

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

**Author(s)**

Barry Grant

**References**

- Grant, B.J. et al. (2006) *Bioinformatics* **22**, 2695–2696.  
Shannon (1948) *The System Technical J.* **27**, 379–422.  
Mirny and Shakhnovich (1999) *J. Mol. Biol.* **291**, 177–196.

**See Also**

[consensus](#), [read.fasta](#)

**Examples**

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

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

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

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

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

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

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

# Plot2: residue frequencies
par(mar = c(5, 4, 0, 2))
image(x=1:ncol(con$freq),
      y=1:nrow(con$freq),
      z=as.matrix(rev(as.data.frame(t(con$freq))))),
      col=col, yaxt="n", xaxt="n",
      xlab="Alignment Position", ylab="Residue Type")
axis(side=1, at=seq(0,length(con$seq),by=5))
axis(side=2, at=c(1:22), labels=aa)
axis(side=3, at=c(1:length(con$seq)), labels =con$seq)
axis(side=4, at=c(1:22), labels=aa)
grid(length(con$seq), length(aa))
box()
```

```

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

par(oldpar)

```

---

example.data

*Bio3d Example Data*

---

## Description

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

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

## Usage

```

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

```

## Format

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

1. `pdbs` is a list of class `pdbs` containing aligned PDB structure data. In the case of transducin this is the output of running `pdbaln` on a set of 53 G[alpha]i structures from the PDB database (see `pdbs$id` or annotation described below for details). The coordinates are fitted onto the first structure based on "core" positions obtained from `core.find` and superposed using the function `pdffit`.
2. `core` is a list of class "core" obtained by running the function `core.find` on the `pdbs` object as described above.
3. `annotation` is a character matrix describing the nucleotide state and bound ligand species for each structure in `pdbs` as obtained from the function `pdb.annotate`.

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



## Source

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

## References

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

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

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

---

filter.cmap

*Contact Map Consensus Filtering*

---

## Description

This function filters a tridimensional contact matrix ( $N \times N \times Z$ ), where  $N$  is the residue number and  $Z$  is the simulation number) selecting only contacts present in at least  $P$  simulations.

## Usage

```
filter.cmap(cm, cutoff.sims = NULL)
```

## Arguments

- |             |   |
|-------------|---|
| cm          | An array of dimensions $N \times N \times Z$ or a list of $N \times N$ matrices containing binary contact values as obtained from <code>cmap</code> . Here, ‘ $N$ ’ is the residue number and ‘ $Z$ ’ the simulation number. The matrix elements should be 1 if two residues are in contact and 0 if they are not in contact. |
| cutoff.sims | A single element numeric vector corresponding to the minimum number of simulations a contact between two residues must be present. If not, it will be set to 0 in the output matrix.  |

## Value

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

## See Also

[cmap](#), [plot.cmap](#)

## Examples

```
## Not run:
  ## load example data
  pdbfile <- system.file("examples/hivp.pdb", package="bio3d")
  pdb <- read.pdb(pdbfile)

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

  ## split the trj example in two
  num.of.frames <- dim(trj)[1]
  trj1 <- trj[1:(num.of.frames/2),]
  trj2 <- trj[((num.of.frames/2)+1):num.of.frames,]

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

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

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

  ## plot the result
  par(pty="s", mfcol=c(1,3))
  plot.cmap(cms[[1]])
  plot.cmap(cms[[2]])
  plot.cmap(cm.filter)

## End(Not run)
```

---

filter.dccm

*Filter for Cross-correlation Matrices (Cij)*

---

## Description

This function builds various cij matrix for correlation network analysis

## Usage

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

**Arguments**

<code>x</code>	A matrix ( $n \times n$ ), a numeric array with 3 dimensions ( $n \times n \times m$ ), a list with $m$ cells each containing $n \times n$ matrix, or a list with 'all.dccm' component, containing atomic correlation values, where "n" is the number of residues and "m" the number of calculations. The matrix elements should be in between -1 and 1. See 'dccm' function in bio3d package for further details.
<code>cutoff.cij</code>	Threshold for each individual correlation value. If NULL, a guessed value will be used. See below for details.
<code>cmap</code>	logical or numerical matrix indicating the contact map. If logical and TRUE, contact map will be calculated with input xyz.
<code>xyz</code>	XYZ coordinates, or a 'pdbs' object obtained from <code>pdbaln</code> or <code>read.fasta.pdb</code> , for contact map calculations.
<code>fac</code>	factor indicating distinct categories of input correlation matrices.
<code>cutoff.sims</code>	Threshold for the number of simulations with observed correlation value above <code>cutoff.cij</code> for the same residue/atomic pairs. See below for details.
<code>collapse</code>	logical, if TRUE the mean matrix will be returned.
<code>extra.filter</code>	Filter to apply in addition to the model chosen.
<code>...</code>	extra arguments passed to function <code>cmap</code> .

**Details**

If `cmap` is TRUE or provided a numerical matrix, the function inspects a set of cross-correlation matrices, or DCCM, and decides edges for correlation network analysis based on:

1.  $\min(\text{abs}(cij)) \geq \text{cutoff.cij}$ , or 2.  $\max(\text{abs}(cij)) \geq \text{cutoff.cij}$  && residues contact each other based on results from `cmap`.

Otherwise, the function filters DCCMs with `cutoff.cij` and return the mean of correlations present in at least `cutoff.sims` calculated matrices.

An internally guessed `cuoff.cij` is used if `cutoff.cij=NULL` is provided. By default, the cutoff is determined by keeping 5% of all residue pairs connected.

**Value**

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

**Author(s)**

Xin-Qiu Yao, Guido Scarabelli & Barry Grant

**References**

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

**See Also**

[cna](#), [dccm](#), [dccm.nma](#), [dccm.xyz](#), [cmap](#), [plot.dccm](#)

**Examples**

```

## Not run:

# Example of transducin
attach(transducin)

gaps.pos <- gap.inspect(pdb$xyz)
modes <- nma.pdb(pdb, ncore=NULL)
dccms <- dccm.enma(modes, ncore=NULL)

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

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

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

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

# NMA on all structures
modes <- nma.pdb(pdb, ncore=NULL)

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

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

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

detach(transducin)

## End(Not run)

```

---

filter.identity

*Percent Identity Filter*


---

**Description**

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

**Usage**

```
filter.identity(aln = NULL, ide = NULL, cutoff = 0.6, verbose = TRUE, ...)
```

**Arguments**

aln	sequence alignment list, obtained from <a href="#">seqaln</a> or <a href="#">read.fasta</a> , or an alignment character matrix. Not used if 'ide' is given.
ide	an optional identity matrix obtained from <a href="#">seqidentity</a> .
cutoff	a numeric identity cutoff value ranging between 0 and 1.
verbose	logical, if TRUE print details of the clustering process.
...	additional arguments passed to and from functions.

**Details**

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

**Value**

Returns a list object with components:

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

**Author(s)**

Barry Grant

**References**

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

**See Also**

[read.fasta](#), [seqaln](#), [seqidentity](#), [entropy](#), [consensus](#)

**Examples**

```
attach(kinesin)

ide.mat <- seqidentity(pdb)

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

```

    main="Sequence Identity", xlab="Identity")

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

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

```

---

filter.rmsd

*RMSD Filter*


---

### Description

Identify and filter subsets of conformations at a given RMSD cutoff.

### Usage

```

filter.rmsd(xyz = NULL, rmsd.mat = NULL, cutoff = 0.5,
           fit = TRUE, verbose = TRUE, inds = NULL, method = "complete",
           ...)

```

### Arguments

xyz	a numeric matrix or list object containing multiple coordinates for pairwise comparison, such as that obtained from <a href="#">read.fasta.pdb</a> . Not used if rmsd.mat is given.
rmsd.mat	an optional matrix of RMSD values obtained from <a href="#">rmsd</a> .
cutoff	a numeric rmsd cutoff value.
fit	logical, if TRUE coordinate superposition is performed prior to RMSD calculation.
verbose	logical, if TRUE progress details are printed.
inds	a vector of indices that selects the elements of xyz upon which the calculation should be based. By default, all the non-gap sites in xyz.
method	the agglomeration method to be used. See function <a href="#">hclust</a> for more information.
...	additional arguments passed to and from functions.

### 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:  
attach(kinesin)  
  
k <- filter.rmsd(xyz=pdbs,cutoff=0.5)  
pdbs$id[k$ind]  
hclustplot(k$tree, h=0.5, ylab="RMSD")  
abline(h=0.5, col="gray")  
  
detach(kinesin)  
  
## End(Not run)
```

---

fit.xyz

*Coordinate Superposition*

---

**Description**

Coordinate superposition with the Kabsch algorithm.

**Usage**

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

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

**Arguments**

<code>fixed</code>	numeric vector of xyz coordinates.
<code>mobile</code>	numeric vector, numeric matrix, or an object with an xyz component containing one or more coordinate sets.
<code>fixed.inds</code>	a vector of indices that selects the elements of <code>fixed</code> upon which fitting should be based.
<code>mobile.inds</code>	a vector of indices that selects the elements of <code>mobile</code> upon which fitting should be based.
<code>full.pdbs</code>	logical, if TRUE “full” coordinate files (i.e. all atoms) are written to the location specified by <code>outpath</code> .
<code>prefix</code>	prefix to <code>mobile\$id</code> to locate “full” input PDB files. Only required if <code>full.pdbs</code> is TRUE.
<code>pdbext</code>	the file name extension of the input PDB files.
<code>outpath</code>	character string specifying the output directory when <code>full.pdbs</code> is TRUE.
<code>xx</code>	numeric vector corresponding to the moving ‘subject’ coordinate set.
<code>yy</code>	numeric vector corresponding to the fixed ‘target’ coordinate set.
<code>xfit</code>	logical vector with the same length as <code>xx</code> , with TRUE elements corresponding to the subset of positions upon which fitting is to be performed.
<code>yfit</code>	logical vector with the same length as <code>yy</code> , with TRUE elements corresponding to the subset of positions upon which fitting is to be performed.
<code>verbose</code>	logical, if TRUE more details are printed.
<code>...</code>	other parameters for <a href="#">read.pdb</a> .
<code>ncore</code>	number of CPU cores used to do the calculation. <code>ncore&gt;1</code> requires package ‘parallel’ installed.
<code>nseg.scale</code>	split input data into specified number of segments prior to running multiple core calculation.



## Details

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

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

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

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

## Value

Returns moved coordinates.

## Author(s)

Barry Grant with `rot.lsqr` 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

```
# PDB server connection required - testing excluded

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

gaps <- gap.inspect(pdbs$xyz)

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

```

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

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

## End(Not run)

##--- Fit two PDBs
A <- read.pdb("1bg2")
A.ind <- atom.select(A, resno=c(256:269), eley='CA')

B <- read.pdb("2kin")
B.ind <- atom.select(B, resno=c(257:270), eley='CA')

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

## Not run:
# Write out moved PDB
C <- B; C$xyz = xyz
write.pdb(pdb=C, file = "moved.pdb")

## End(Not run)

```

---

fluct.nma

*NMA Fluctuations*


---

### Description

Calculates the atomic fluctuations from normal modes analysis.

### Usage

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

### Arguments

nma	a list object of class "nma" (obtained with <a href="#">nma</a> ).
mode.inds	a numeric vector containing the the mode numbers in which the calculation should be based.

**Details**

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

See examples for more details.

**Value**

Returns a numeric vector of atomic fluctuations.

**Author(s)**

Lars Skjaerven

**References**

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

**See Also**

[nma](#)

**Examples**

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

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

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

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

---

formula2mass

*Chemical Formula to Mass Converter*

---

**Description**

Compute the molar mass associated to a chemical formula.

**Usage**

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

**Arguments**

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

**Details**

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

**Value**

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

**Author(s)**

Lars Skjaerven

**See Also**

[atom2ele](#), [atom2mass](#)

**Examples**

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

---

gap.inspect

*Alignment Gap Summary*

---

**Description**

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

**Usage**

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

**Arguments**

x                        a matrix or an alignment data structure obtained from [read.fasta](#) or [read.fasta.pdb](#).

**Details**

Reports the number of gap characters per row (i.e. sequence) and per column (i.e. position) for a given alignment. In addition, the indices for gap and non-gap containing columns 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 columns
f.inds	indices for non-gap containing columns
bin	a binary numeric matrix with the same dimensions as the alignment, with 0 at non-gap positions and 1 at gap positions.

**Note**

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

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

**Author(s)**

Barry Grant

**References**

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

**See Also**

[read.fasta](#), [read.fasta.pdb](#)

**Examples**

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

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

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

---

 geostas

*GeoStaS Domain Finder*


---

## Description

Identifies geometrically stable domains in biomolecules

## Usage

```
geostas(...)

## Default S3 method:
geostas(...)

## S3 method for class 'xyz'
geostas(xyz, amsm = NULL, k = 3, pairwise = TRUE,
        clustalg = "kmeans", fit = TRUE, ncore = NULL, verbose=TRUE, ...)

## S3 method for class 'nma'
geostas(nma, m.inds = 7:11, verbose=TRUE, ...)

## S3 method for class 'enma'
geostas(enma, pdbs = NULL, m.inds = 1:5, verbose=TRUE, ...)

## S3 method for class 'pdb'
geostas(pdb, inds = NULL, verbose=TRUE, ...)

## S3 method for class 'pdbs'
geostas(pdbs, verbose=TRUE, ...)

amsm.xyz(xyz, ncore = NULL)

## S3 method for class 'geostas'
print(x, ...)
```

## Arguments

...	arguments passed to and from functions, such as <a href="#">kmeans</a> , and <a href="#">hclust</a> which are called internally in <code>geostas.xyz</code> .
xyz	numeric matrix of xyz coordinates as obtained e.g. by <a href="#">read.ncdf</a> , <a href="#">read.dcd</a> , or <a href="#">mktrj</a> .
amsm	a numeric matrix as obtained by <a href="#">amsm.xyz</a> (convenient e.g. for re-doing only the clustering analysis of the 'AMSM' matrix).
k	an integer scalar or vector with the desired number of groups.
pairwise	logical, if TRUE use pairwise clustering of the atomic movement similarity matrix (AMSM), else columnwise.

<code>clustalg</code>	a character string specifying the clustering algorithm. Allowed values are ‘kmeans’ and ‘hclust’.
<code>fit</code>	logical, if TRUE coordinate superposition on identified core atoms is performed prior to the calculation of the AMS matrix.
<code>ncore</code>	number of CPU cores used to do the calculation. <code>ncore&gt;1</code> requires package ‘parallel’ installed.
<code>verbose</code>	logical, if TRUE details of the geostas calculations are printed to screen.
<code>nma</code>	an ‘nma’ object as obtained from function <code>nma</code> . Function <code>mktrj</code> is used internally to generate a trajectory based on the normal modes.
<code>m.inds</code>	the mode number(s) along which trajectory should be made (see function <code>mktrj</code> ).
<code>enma</code>	an ‘enma’ object as obtained from function <code>nma.pdbs</code> . Function <code>mktrj</code> is used internally to generate a trajectory based on the normal modes.
<code>pdbs</code>	a ‘pdbs’ object as obtained from function <code>pdbaln</code> or <code>read.fasta.pdb</code> .
<code>pdb</code>	a ‘pdb’ object as obtained from function <code>read.pdb</code> .
<code>inds</code>	a ‘select’ object as obtained from function <code>atom.select</code> giving the atomic indices at which the calculation should be based. By default the function will attempt to locate C-alpha atoms using function <code>atom.select</code> .
<code>x</code>	a ‘geostas’ object as obtained from function <code>geostas</code> .

## Details

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

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

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

See examples for more details.

## Value

Returns a list object of type ‘geostas’ with the following components:

<code>amsm</code>	a numeric matrix of atomic movement similarity (AMSM).
<code>fit.inds</code>	a numeric vector of xyz indices used for fitting.
<code>grps</code>	a numeric vector containing the domain assignment per residue.
<code>atomgrps</code>	a numeric vector containing the domain assignment per atom (only provided for <code>geostas.pdb</code> ).

inds            a list of atom 'select' objects with indices to corresponding to the identified domains.

### Note

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

### Author(s)

Julia Romanowska and Lars Skjaerven

### References

Romanowska, J. et al. (2012) *JCTC* **8**, 2588–2599. Skjaerven, L. et al. (2014) *BMC Bioinformatics* **15**, 399. Grant, B.J. et al. (2006) *Bioinformatics* **22**, 2695–2696.

### See Also

[plot.geostas](#), [read.pdb](#), [mktrj](#), [read.ncdf](#), [read.dcd](#), [nma](#), [dccm](#).

### Examples

```
# PDB server connection required - testing excluded

#### NMR-ensemble example
## Read a multi-model PDB file
pdb <- read.pdb("1d1d", multi=TRUE)

## Find domains and write PDB
gs <- geostas(pdb, fit=TRUE)

## Plot a atomic movement similarity matrix
plot.geostas(gs, contour=FALSE)

## Fit all frames to the 'first' domain
domain.inds <- gs$inds[[1]]

xyz <- pdbfit(pdb, inds=domain.inds)

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

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

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



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

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

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

#### Trajectory example
## Read inn DCD trajectory file, fit coordinates
dcdfile <- system.file("examples/hivp.dcd", package = "bio3d")
trj <- read.dcd(dcdfile)
xyz <- fit.xyz(trj[1,], trj)

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

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

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

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

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

# Align structures
pdb <- pdbaln(files)

# Find domains
gs <- geostas(pdb, k=4, fit=TRUE)

# Superimpose to core region
pdb$xyz <- pdbfit(pdb, inds=gs$fit.inds)

# Principal component analysis
pc.xray <- pca(pdb)

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

```

##- Same, but more manual approach
gaps.pos <- gap.inspect(pdb$xyz)

# Find core region
core <- core.find(pdb)

# Fit to core region
xyz <- fit.xyz(pdb$xyz[1, gaps.pos$f.inds],
              pdb$xyz[, gaps.pos$f.inds],
              fixed.inds=core$xyz,
              mobile.inds=core$xyz)

# Find domains
gs <- geostas(xyz, k=4, fit=FALSE)

# Perform PCA
pc.xray <- pca.xyz(xyz)

# Make trajectory
mktrj(pc.xray, pc=1, chain=gs$grps)

## End(Not run)

```

---

get.pdb

*Download PDB Coordinate Files*


---

## Description

Downloads PDB coordinate files from the RCSB Protein Data Bank.

## Usage

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

## Arguments

ids	A character vector of one or more 4-letter PDB codes/identifiers or 6-letter PDB-ID_Chain-ID of the files to be downloaded, or a 'blast' object containing 'pdb.id'.
path	The destination path/directory where files are to be written.
URLonly	logical, if TRUE a character vector containing the URL path to the online file is returned and files are not downloaded. If FALSE the files are downloaded.
overwrite	logical, if FALSE the file will not be downloaded if it already exist.
gzip	logical, if TRUE the gzipped PDB will be downloaded and extracted locally.

split	logical, if TRUE <code>pdbsplit</code> function will be called to split pdb files into separated chains.
format	format of the data file: 'pdb' or 'cif' for PDB and mmCIF file formats, respectively.
verbose	print details of the reading process.
ncore	number of CPU cores used to do the calculation. <code>ncore&gt;1</code> requires package 'parallel' installed.
...	extra arguments passed to <code>pdbsplit</code> function.

### Details

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

### Value

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

### Author(s)

Barry Grant

### References

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

For a description of PDB format (version3.3) see:

<http://www.wwpdb.org/documentation/format33/v3.3.html>.

### See Also

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

### Examples

```
# PDB server connection required - testing excluded

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

## These URLs can be used by 'read.pdb'
pdb <- read.pdb( get.pdb("5p21", URL=TRUE) )
summary(pdb)

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

---

`get.seq`*Download FASTA Sequence Files*

---

**Description**

Downloads FASTA sequence files from the NCBI nr, SWISSPROT/UNIPROT, OR RCSB PDB databases.

**Usage**

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

**Arguments**

<code>ids</code>	A character vector of one or more appropriate database codes/identifiers of the files to be downloaded.
<code>outfile</code>	A single element character vector specifying the name of the local file to which sequences will be written.
<code>db</code>	A single element character vector specifying the database from which sequences are to be obtained.
<code>verbose</code>	logical, if TRUE URL details of the download process are printed.

**Details**

This is a basic function to automate sequence file download from the databases including NCBI nr, SWISSPROT/UNIPROT, and RCSB PDB.

**Value**

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

<code>ali</code>	an alignment character matrix with a row per sequence and a column per equivalent aminoacid/nucleotide.
<code>ids</code>	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: <https://www.ncbi.nlm.nih.gov/BLAST/blastcgihelp.shtml>. When reading alignment files, the dash '-' is interpreted as the gap character.

**Author(s)**

Barry Grant

## References

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

## See Also

[blast.pdb](#), [read.fasta](#), [read.fasta.pdb](#), [get.pdb](#)

## Examples

```
## Not run:
## Sequence identifiers (GI or PDB codes e.g. from blast.pdb etc.)
get.seq( c("P01112", "Q61411", "P20171") )

##aa <-get.seq( c("4q21", "5p21") )
##aa$id
##aa$ali

## End(Not run)
```

---

gnm

*Gaussian Network Model*

---

## Description

Perform Gaussian network model (GNM) based normal mode analysis (NMA) for a protein structure.

## Usage

```
gnm(x, ...)
```

```
## S3 method for class 'pdb'
gnm(x, inds = NULL, temp = 300, keep = NULL,
     outmodes = NULL, gamma = 1, cutoff = 8, check.connect = TRUE, ...)
```

```
## S3 method for class 'pdbs'
gnm(x, fit = TRUE, full = FALSE, subspace = NULL,
     rm.gaps = TRUE, gc.first = TRUE, ncore = NULL, ...)
```

## Arguments

x	an object of class <code>pdb</code> as obtained from function <a href="#">read.pdb</a> .
...	(in <code>gnm.pdbs</code> ) additional arguments passed to <code>gnm.pdb</code> .
inds	atom and xyz coordinate indices obtained from <a href="#">atom.select</a> that selects the elements of <code>pdb</code> upon which the calculation should be based. If not provided the function will attempt to select all calpha atoms automatically.

temp	numerical, temperature for which the amplitudes for scaling the atomic displacement vectors are calculated. Set 'temp=NULL' to avoid scaling.
keep	numerical, final number of modes to be stored. Note that all subsequent analyses are limited to this subset of modes. This option is useful for very large structures and cases where memory may be limited.
outmodes	atom indices as obtained from <code>atom.select</code> specifying the atoms to include in the resulting mode object.
gamma	numerical, global scale of the force constant.
cutoff	numerical, distance cutoff for pair-wise interactions.
check.connect	logical, if TRUE check chain connectivity.
fit	logical, if TRUE C-alpha coordinate based superposition is performed prior to normal mode calculations.
full	logical, if TRUE return the complete, full structure, 'nma' objects.
subspace	number of eigenvectors to store for further analysis.
rm.gaps	logical, if TRUE obtain the hessian matrices for only atoms in the aligned positions (non-gap positions in all aligned structures). Thus, gap positions are removed from output.
gc.first	logical, if TRUE will call <code>gc()</code> first before mode calculation for each structure. This is to avoid memory overload when <code>ncore &gt; 1</code> .
ncore	number of CPU cores used to do the calculation.

### Details

This function builds a Gaussian network model (an isotropic elastic network model) for C-alpha atoms and performs subsequent normal mode analysis (NMA). The model employs a distance cutoff for the network construction: Atom pairs with distance falling within the cutoff have a harmonic interaction with a uniform force constant; Otherwise atoms have no interaction. Output contains N-1 (N, the number of residues) non-trivial modes (i.e. the degree of freedom is N-1), which can then be used to calculate atomic fluctuations and covariance.

### Value

Returns an object of class 'gnm' with the following components:

force.constants	numeric vector containing the force constants corresponding to each mode.
fluctuations	numeric vector of atomic fluctuations.
U	numeric matrix with columns containing the raw eigenvectors.
L	numeric vector containing the raw eigenvalues.
xyz	numeric matrix of class xyz containing the Cartesian coordinates in which the calculation was performed.
temp	numerical, temperature for which the amplitudes for scaling the atomic displacement vectors are calculated.
triv.modes	number of trivial modes.
natoms	number of C-alpha atoms.
call	the matched call.

**Author(s)**

Xin-Qiu Yao & Lars Skjaerven

**References**

Bahar, I. et al. (1997) *Folding Des.* **2**, 173.

**See Also**

[gnm.pdbs](#)

**Examples**

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

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

## Print modes
print(modes)

## Plot modes
plot(modes)
```

---

hclustplot

*Dendrogram with Clustering Annotation*

---

**Description**

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

**Usage**

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

**Arguments**

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

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

### Details

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

See examples for further details on usage.

### Value

Called for its effect.

### Note

Argument 'horiz=TRUE' currently not supported.

### Author(s)

Lars Skjaerven

### References

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

### See Also

[plot.hclust](#), [plot.dendrogram](#), [hclust](#), [cutree](#).

### Examples

```
# Redundant testing excluded

attach(transducin)

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



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

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

detach(transducin)
```

---

hmmmer

*HMMER Sequence Search*


---

### Description

Perform a HMMER search against the PDB, NR, swissprot or other sequence and structure databases.

### Usage

```
hmmmer(seq, type="phmmmer", db = NULL, verbose = TRUE, timeout = 90)
```

### Arguments

seq	a multi-element character vector containing the query sequence. Alternatively a 'fasta' object as obtained from functions <code>get.seq</code> or <code>read.fasta</code> can be provided.
type	character string specifying the 'HMMER' job type. Current options are 'phmmmer', 'hmmscan', 'hmmsearch', and 'jackhmmmer'.
db	character string specifying the database to search. Current options are 'pdb', 'nr', 'swissprot', 'pfam', etc. See 'details' for a complete list.
verbose	logical, if TRUE details of the download process is printed.
timeout	integer specifying the number of seconds to wait for the blast reply before a time out occurs.

### Details

This function employs direct HTTP-encoded requests to the HMMER web server. HMMER can be used to search sequence databases for homologous protein sequences. The HMMER server implements methods using probabilistic models called profile hidden Markov models (profile HMMs).

There are currently four types of HMMER search to perform:

- 'phmmmer': protein sequence vs protein sequence database. (input argument `seq` must be a sequence).

Allowed options for `type` includes: 'env\_nr', 'nr', 'refseq', 'pdb', 'rp15', 'rp35', 'rp55', 'rp75', 'swissprot', 'unimes', 'uniprotkb', 'uniprotrefprot', 'pfamseq'.

- 'hmmsearch': protein sequence vs profile-HMM database.  
(input argument seq must be a sequence).

Allowed options for type includes: 'pfam', 'gene3d', 'superfamily', 'tigrfam'.

- 'hmmsearch': protein alignment/profile-HMM vs protein sequence database.  
(input argument seq must be an alignment).

Allowed options for type includes: 'pdb', 'swissprot'.

- 'jackhmmmer': iterative search vs protein sequence database.  
(input argument seq must be an alignment). 'jackhmmmer' functionality incomplete!!

Allowed options for type includes: 'env\_nr', 'nr', 'refseq', 'pdb', 'rp15', 'rp35', 'rp55', 'rp75', 'swissprot', 'unimes', 'uniprotkb', 'uniprotrefprot', 'pfamseq'.

More information can be found at the HMMER website:

<http://hmmmer.org>

## Value

A list object with components 'hit.tbl' and 'url'. 'hit.tbl' is a data frame with multiple components depending on the selected job 'type'. Frequently reported fields include:

name	a character vector containing the name of the target.
acc	a character vector containing the accession identifier of the target.
acc2	a character vector containing secondary accession of the target.
pdb.id	same as 'acc'.
id	a character vector containing Identifier of the target
desc	a character vector containing entry description.
score	a numeric vector containing bit score of the sequence (all domains, without correction).
bitscore	same as 'score'.
pvalue	a numeric vector containing the P-value of the score.
evaluate	a numeric vector containing the E-value of the score.
mlog.evaluate	a numeric vector containing minus the natural log of the E-value.
nregions	a numeric vector containing Number of regions evaluated.
nvelopes	a numeric vector containing the number of envelopes handed over for domain definition, null2, alignment, and scoring.
ndom	a numeric vector containing the total number of domains identified in this sequence.
nreported	a numeric vector containing the number of domains satisfying reporting thresholding.
nincluded	a numeric vector containing the number of domains satisfying inclusion thresholding.
taxid	a character vector containing The NCBI taxonomy identifier of the target (if applicable).
species	a character vector containing the species name.

kg a character vector containing the kingdom of life that the target belongs to - based on placing in the NCBI taxonomy tree.

More details can be found at the HMMER website:  
<https://www.ebi.ac.uk/Tools/hmmmer/help/api>

### Note

Note that the chained 'pdb' HMMER field (used for redundant PDBs) is included directly into the result list (applies only when db='pdb'). In this case, the 'name' component of the target contains the parent (non redundant) entry, and the 'acc' component the chained PDB identifiers. The search results will therefore provide duplicated PDB identifiers for component \$name, while \$acc should be unique.

### Note

Online access is required to query HMMER services.

### Author(s)

Lars Skjaerven

### References

Grant, B.J. et al. (2006) *Bioinformatics* **22**, 2695–2696.  
Finn, R.D. et al. (2011) *Nucl. Acids Res.* **39**, 29–37. Eddy, S.R. (2011) *PLoS Comput Biol* **7**(10): e1002195.  
See also the 'HMMER' website:  
<http://hmmmer.org>

### See Also

[blast.pdb](#), [plot.blast](#), [seqaln](#), [get.seq](#), [pfam](#), [uniprot](#)

### Examples

```
## Not run:  
# HMMER server connection required - testing excluded  
  
##- PHMMER  
seq <- get.seq("2abl_A", outfile=tempfile())  
res <- hmmmer(seq, db="pdb")  
  
##- HMMSCAN  
fam <- hmmmer(seq, type="hmmscan", db="pfam")  
pfam.aln <- pfam(fam$hit.tbl$acc[1])  
  
##- HMMSEARCH  
hmm <- hmmmer(pfam.aln, type="hmmsearch", db="pdb")  
unique(hmm$hit.tbl$species)  
hmm$hit.tbl$acc
```

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

---

```
identify.cna
```

```
Identify Points in a CNA Protein Structure Network Plot
```

---

### Description

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

### Usage

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

### Arguments

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

### Details

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

### Value

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

### Author(s)

Guido Scarabelli and Barry Grant

**See Also**

[plot.cna](#), [identify](#), [plot.igraph](#), [plot.communities](#), [igraph.plotting](#)

**Examples**

```
## Not run:

if (!requireNamespace("igraph", quietly = TRUE)) {
  message('Need igraph installed to run this example')
} else {

  attach(hivp)

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

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

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

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

  detach(hivp)

}

## End(Not run)
```

---

inner.prod

*Mass-weighted Inner Product*

---

**Description**

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

**Usage**

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

**Arguments**

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

## Details

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

See examples for more details.

## Value

Returns the inner product(s).

## Author(s)

Lars Skjaerven

## References

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

## See Also

[nma](#), [normalize.vector](#)

## Examples

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

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

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

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

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

---

inspect.connectivity    *Check the Connectivity of Protein Structures*

---

### Description

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

### Usage

```
inspect.connectivity(pdb, cut=4.)
```

### Arguments

pdbs	an object of class <code>3dalign</code> as obtained from function <code>pdbaln</code> or <code>read.fasta.pdb</code> ; a xyz matrix containing the cartesian coordinates of C-alpha atoms; or a 'pdb' object as obtained from function <code>read.pdb</code> .
cut	cutoff value to determine residue connectivity.

### Details

Utility function for checking if the PDB structures in a 'pdbs' object contains missing residues inside the structure.

### Value

Returns a vector.

### Author(s)

Lars Skjaerven

### References

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

### See Also

[dm](#), [gap.inspect](#)

### Examples

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

## Sequence Alignment, and connectivity check
pdbs <- pdbaln(files)
```

```
cons <- inspect.connectivity(pdbbs)

## omit files with missing residues
files = files[cons]

## End(Not run)
```

---

is.gap

*Gap Characters*

---

### Description

Test for the presence of gap characters.

### Usage

```
is.gap(x, gap.char = c("-", "."))
```

### Arguments

**x** an R object to be tested. Typically a sequence vector or sequence/structure alignment object as returned from `seqaln`, `pdbaln` etc.

**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 vector, or the same length as the number of columns present in an alignment input object 'x'. In the later case TRUE elements corresponding to 'gap.char' matches in any alignment column (i.e. gap containing columns).

### 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](#), [pdbaln](#)

**Examples**

```
is.gap( c("G",".", "X", "-", "G", "K", "S", "T") )

## Not run:
aln <- read.fasta( system.file("examples/kif1a.fa",
                             package = "bio3d") )

##- Print only non-gap positions (i.e. no gaps in any sequence)
aln$ali[, !is.gap(aln) ]

##- Mask any existing gaps with an "X"
xaln <- aln
xaln$ali[ is.gap(xaln$ali) ]="X"

##- Read a new PDB and align its sequence to the existing masked alignment
pdb <- read.pdb( "1mkj" )
seq2aln(pdbseq(pdb), xaln, id = "1mkj")

## End(Not run)
```

---

is.mol2

*Is an Object of Class 'mol2'?*

---

**Description**

Checks whether its argument is an object of class 'mol2'.

**Usage**

```
is.mol2(x)
```

**Arguments**

x                    an R object.

**Details**

Tests if the object 'x' is of class 'mol2' (is.mol2), i.e. if 'x' has a "class" attribute equal to mol2.

**Value**

TRUE if x is an object of class 'mol2' and FALSE otherwise

**See Also**[read.mol2](#)**Examples**

```
# Read a PDB file
mol <- read.mol2( system.file("examples/aspirin.mol2", package="bio3d") )
is.mol2(mol)
```

---

is.pdb

*Is an Object of Class 'pdb(s)'?*

---

**Description**

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

**Usage**

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

**Arguments**

x                    an R object.

**Details**

Tests if the object 'x' is of class 'pdb' (is.pdb) or 'pdbs' (is.pdbs), i.e. if 'x' has a "class" attribute equal to pdb or pdbs.

**Value**

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

**See Also**

[read.pdb](#), [read.fasta.pdb](#), [pdbaln](#)

**Examples**

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

---

`is.select`*Is an Object of Class 'select'?*

---

**Description**

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

**Usage**

```
is.select(x)
```

**Arguments**

`x` an R object to be tested.

**Details**

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

**Value**

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

**Author(s)**

Julien Ide

**See Also**

[atom.select](#)

**Examples**

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

# Print structure summary
atom.select(pdb)

# Select all C-alpha atoms with residues numbers between 43 and 54
ca.inds <- atom.select(pdb, "calpha", resno=43:54)
is.select(ca.inds)
```

is.xyz

*Is an Object of Class 'xyz'?*

---

**Description**

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

**Usage**

```
is.xyz(x)
as.xyz(x)
```

**Arguments**

x                    an R object to be tested

**Details**

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

**Value**

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

**See Also**

[read.pdb](#), [read.ncdf](#), [read.dcd](#), [fit.xyz](#)

**Examples**

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

---

layout.cna*Protein Structure Network Layout*

---

**Description**

Determine protein structure network layout in 2D and 3D from the geometric center of each community.

**Usage**

```
layout.cna(x, pdb, renumber=TRUE, k=2, full=FALSE)
```

**Arguments**

x	A protein structure network object as obtained from the 'cna' function.
pdb	A pdb class object as obtained from the 'read.pdb' function.
renumber	Logical, if TRUE the input 'pdb' will be re-numbered starting at residue number one before community coordinate averages are calculated.
k	A single element numeric vector between 1 and 3 specifying the returned coordinate dimensions.
full	Logical, if TRUE the full all-Calpha atom network coordinates will be returned rather than the default clustered network community coordinates.

**Details**

This function calculates the geometric center for each community from the atomic position of its Calpha atoms taken from a corresponding PDB file. Care needs to be taken to ensure the PDB residue numbers and the community vector names/length match.

The community residue membership are typically taken from the input network object but can be supplied as a list object with 'x\$communities\$membership'.

**Value**

A numeric matrix of Nxk, where N is the number of communities and k the number of dimensions requested.

**Author(s)**

Guido Scarabelli and Barry Grant

**See Also**

[plot.cna](#), [plot.communities](#), [igraph.plotting](#), [plot.igraph](#)

**Examples**

```
if (!requireNamespace("igraph", quietly = TRUE)) {
  message('Need igraph installed to run this example')
} else {

# Load the correlation network
attach(hivp)

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

# Plot will be slow
#xy <- plot.cna(net)
#plot3d.cna(net, pdb)
```

```
layout.cna(net, pdb, k=3)
layout.cna(net, pdb)

# can be used as input to plot.cna and plot3d.cna...
# plot.cna( net, layout=layout.cna(net, pdb) )
# plot3d.cna(net, pdb, layout=layout.cna(net, pdb, k=3))

detach(hivp)

}
```

---

lbio3d

*List all Functions in the bio3d Package*

---

### **Description**

A simple shortcut for `ls("package:bio3d")`.

### **Usage**

```
lbio3d()
```

### **Value**

A character vector of function names from the bio3d package.

### **Author(s)**

Barry Grant

### **References**

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

---

load.enmff

*ENM Force Field Loader*

---

### **Description**

Load force field for elastic network normal mode calculation.

**Usage**

```

load.enmff(ff = 'calpha')
ff.calpha(r, rmin=2.9, ...)
ff.anm(r, cutoff=15, gamma=1, ...)
ff.pfanm(r, cutoff=NULL, ...)
ff.sdenm(r, atom.id, pdb, ...)
ff.reach(r, atom.id, ...)
ff.aenm(r, ...)
ff.aenm2(r, atom.id, pdb, ...)

```

**Arguments**

ff	a character string specifying the force field to use: 'calpha', 'anm', 'pfanm', 'reach', or 'sdenm'.
r	a numeric vector of c-alpha distances.
rmin	lowest allowed atom-atom distance for the force constant calculation. The default of 2.9A is based on an evaluation of 24 high-resolution X-ray structures (< 1A).
cutoff	numerical, cutoff for pair-wise interactions.
gamma	numerical, global scaling factor.
atom.id	atomic index.
pdb	a pdb object as obtained from function read.pdb.
...	additional arguments passed to and from functions.

**Details**

This function provides a collection of elastic network model (ENM) force fields for normal modes analysis (NMA) of protein structures. It returns a function for calculating the residue-residue spring force constants.

The 'calpha' force field - originally developed by Konrad Hinsen - is the recommended one for most applications. It employs a spring force constant differentiating between nearest-neighbour pairs along the backbone and all other pairs. The force constant function was parameterized by fitting to a local minimum of a crambin model using the AMBER94 force field.

The implementation of the 'ANM' (Anisotropic Network Model) force field originates from the lab of Ivet Bahar. It uses a simplified (step function) spring force constant based on the pair-wise distance. A variant of this from the Jernigan lab is the so-called 'pfANM' (parameter free ANM) with interactions that fall off with the square of the distance.

The 'sdENM' (by Dehouck and Mikhailov) employs residue specific spring force constants. It has been parameterized through a statistical analysis of a total of 1500 NMR ensembles.

The 'REACH' force field (by Moritsugu and Smith) is parameterized based on variance-covariance matrices obtained from MD simulations. It employs force constants that fall off exponentially with distance for non-bonded pairs.

The all-atom ENM force fields ('aenm' and 'aenm2') was obtained by fitting to a local energy minimum of a crambin model derived from the AMBER99SB force field (same approach as in Hinsen et al 2000). It employs a pair force constant function which falls as  $r^{-6}$ . 'aanma2' employs

additionally specific force constants for covalent and intra-residue atom pairs. See also [aanma](#) for more details.

See references for more details on the individual force fields.

### Value

'load.enmff' returns a function for calculating the spring force constants. The 'ff' functions returns a numeric vector of residue-residue spring force constants.

### Note

The arguments 'atom.id' and 'pdb' are used from within function 'build.hessian' for functions that are not simply a function of the pair-wise distance. e.g. the force constants in the 'sdENM' model computes the force constants based on a function of the residue types and calpha distance.

### Author(s)

Lars Skjaerven

### References

Skjaerven, L. et al. (2014) *BMC Bioinformatics* **15**, 399. Hinsen, K. et al. (2000) *Chemical Physics* **261**, 25–37. Atilgan, A.R. et al. (2001) *Biophysical Journal* **80**, 505–515. Dehouck Y. & Mikhailov A.S. (2013) *PLoS Comput Biol* **9**:e1003209. Moritsugu K. & Smith J.C. (2008) *Biophysical Journal* **95**, 1639–1648. Yang, L. et al. (2009) *PNAS* **104**, 12347–52. Grant, B.J. et al. (2006) *Bioinformatics* **22**, 2695–2696.

### See Also

[nma](#), [build.hessian](#)

### Examples

```
## Load the c-alpha force field
pfc.fun <- load.enmff('calpha')

## Calculate the pair force constant for a set of C-alpha distances
force.constants <- pfc.fun( seq(4,8, by=0.5) )

## Calculate the complete spring force constant matrix
## Fetch PDB
pdb <- read.pdb( system.file("examples/1hel.pdb", package="bio3d") )

## Fetch only c-alpha coordinates
ca.inds <- atom.select(pdb, 'calpha')
xyz <- pdb$xyz[ca.inds$xyz]

## Calculate distance matrix
dists <- dm.xyz(xyz, mask.lower=FALSE)

## all pair-wise spring force constants
```



```
fc.matrix <- apply(dists, 1, pfc.fun)
```

---

mask	<i>Mask a Subset of Atoms in a DCCM Object.</i>
------	---

---

### Description

Produce a new DCCM object with selected atoms masked.

### Usage

```
mask(...)
```

```
## S3 method for class 'dccm'
```

```
mask(dccm, pdb = NULL, a.inds = NULL, b.inds = NULL, ...)
```

### Arguments

dccm	a DCCM structure object obtained from function <a href="#">dccm</a> .
pdb	a PDB structure object obtained from <a href="#">read.pdb</a> . Must match the dimensions of dccm.
a.inds	a numeric vector containing the indices of the elements of the DCCM matrix in which should not be masked. Alternatively, if pdb is provided a selection object (as obtained from <a href="#">atom.select</a> ) can be provided.
b.inds	a numeric vector containing the indices of the elements of the DCCM matrix in which should not be masked.
...	arguments not passed anywhere.

### Details

This is a basic utility function for masking a DCCM object matrix to highlight user-selected regions in the correlation network.

When both a.inds and b.inds are provided only their intersection is retained. When only a.inds is provided then the corresponding region to everything else is retained.

Note: The current version assumes that the input PDB corresponds to the input DCCM. In many cases this will correspond to a PDB object containing only CA atoms.

### Value

Returns a matrix list of class "dccm" with the indices/atoms not corresponding to the selection masked.

### Author(s)

Lars Skjaerven

## References

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

## See Also

[dccm](#), [atom.select](#)

## Examples

```
## Calculate DCCM
pdb <- read.pdb( system.file("examples/1hel.pdb", package="bio3d") )
cij <- dccm(nma(pdb))

## Mask DCCM matrix according to matrix indices
cijm <- mask(cij, a.ind=40:50, b.ind=80:90)
plot(cijm)

## Retain only 40:50 to everything else
cijm <- mask(cij, a.ind=40:50)
plot(cijm)

## Mask DCCM matrix according PDB selection
pdb.ca <- trim(pdb, "calpha")
a.ind <- atom.select(pdb.ca, resno=40:50)
b.ind <- atom.select(pdb.ca, resno=80:90)

# Provide pdb object corresponding to input dccm
cijm <- mask(cij, pdb.ca, a.ind, b.ind)
plot(cijm)
```

---

mktrj

*PCA / NMA Atomic Displacement Trajectory*

---

## Description

Make a trajectory of atomic displacements along a given principal component / normal mode.

## Usage

```
mktrj(...)
```

```
## S3 method for class 'pca'
mktrj(pca = NULL, pc = 1, mag = 1, step = 0.125, file =
NULL, pdb = NULL, rock=TRUE, ...)
```

```
## S3 method for class 'nma'
mktrj(nma = NULL, mode = 7, mag = 10, step = 1.25, file = NULL,
      pdb = NULL, rock=TRUE, ...)
```

```
## S3 method for class 'enma'
mktrj(enma = NULL, pdba = NULL, s.ind = NULL, m.ind = NULL,
      mag = 10, step = 1.25, file = NULL, rock = TRUE, ncore = NULL, ...)
```

### Arguments

pca	an object of class "pca" as obtained with function <a href="#">pca.xyz</a> or <a href="#">pca</a> .
nma	an object of class "nma" as obtained with function <a href="#">nma.pdb</a> .
enma	an object of class "enma" as obtained with function <a href="#">nma.pdba</a> .
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/mode.
file	a character vector giving the output PDB file name.
pdb	an object of class "pdb" as obtained from <a href="#">read.pdb</a> or class "pdbs" as obtained from <a href="#">read.fasta.pdb</a> . If not NULL, used as reference to write the PDB file.
rock	logical, if TRUE the trajectory rocks.
mode	the mode number along which displacements should be made.
pdbs	a list object of class "pdbs" (obtained with <a href="#">pdbaIn</a> or <a href="#">read.fasta.pdb</a> ) which corresponds to the "enma" object.
s.ind	index or indices pointing to the structure(s) in the enma object for which the trajectory shall be generated.
m.ind	the mode number(s) along which displacements should be made.
ncore	number of CPU cores used to do the calculation. ncore>1 requires package 'parallel' installed.
...	additional arguments passed to and from functions (e.g. to function <a href="#">write.pdb</a> ).

### Details

Trajectory frames are built from reconstructed Cartesian coordinates produced by interpolating from the mean structure along a given pc or mode, 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).

### Note

Molecular graphics software such as VMD or PyMOL is useful for viewing trajectories see e.g: <http://www.ks.uiuc.edu/Research/vmd/>.

### Author(s)

Barry Grant, Lars Skjaerven

## References

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

## See Also

[pca](#), [nma](#), [nma.pdbs](#), [pymol.modes](#).

## Examples

```
## Not run:

##- PCA example
attach(transducin)

# Calculate principal components
pc.xray <- pca(pdbs, fit=TRUE)

# Write PC trajectory of pc=1
outfile = tempfile()
a <- mktrj(pc.xray, file = outfile)
outfile

detach(transducin)

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

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

## Visualize modes
outfile = file.path(tempdir(), "mode_7.pdb")
mktrj(modes, mode=7, pdb=pdb, file = outfile)
outfile

## End(Not run)
```

---

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 either a multiple or single element character vector. The dot character and other valid regex 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**

[regexr](#), [read.fasta](#), [pdbseq](#)

**Examples**

```
# PDB server connection required - testing excluded

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

---

mustang

*Structure-based Sequence Alignment with MUSTANG*

---

**Description**

Create a multiple sequence alignment from a bunch of PDB files.

**Usage**

```
mustang(files, exefile="mustang", outfile="aln.mustang.fa",
        cleanpdb=FALSE, cleandir="mustangpds", verbose=TRUE)
```

### Arguments

files	a character vector of PDB file names.
exefile	file path to the 'MUSTANG' program on your system (i.e. how is 'MUSTANG' invoked).
outfile	name of 'FASTA' output file to which alignment should be written.
cleanpdb	logical, if TRUE iterate over the PDB files and map non-standard residues to standard residues (e.g. SEP->SER..) to produce 'clean' PDB files.
cleandir	character string specifying the directory in which the 'clean' PDB files should be written.
verbose	logical, if TRUE 'MUSTANG' warning and error messages are printed.

### Details

Structure-based sequence alignment with 'MUSTANG' attempts to arrange and align the sequences of proteins based on their 3D structure.

This function calls the 'MUSTANG' program, to perform a multiple structure alignment, which MUST BE INSTALLED on your system and in the search path for executables.

Note that non-standard residues are mapped to "Z" in MUSTANG. As a workaround the bio3d 'mustang' function will attempt to map any non-standard residues to standard residues (e.g. SEP->SER, etc). To avoid this behaviour use 'cleanpdb=FALSE'.

### Value

A list with two components:

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

### Note

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

### Author(s)

Lars Skjaerven

### References

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

'MUSTANG' is the work of Konagurthu et al: Konagurthu, A.S. et al. (2006) *Proteins* **64**(3):559–74.

More details of the 'MUSTANG' algorithm, along with download and installation instructions can be obtained from:

<https://lcb.infotech.monash.edu/mustang/>.

**See Also**

[read.fasta](#), [read.fasta.pdb](#), [pdbaln](#), [plot.fasta](#), [seqaln](#)

**Examples**

```
## Not run:

if(!check.utility('mustang')) {
  message('Need MUSTANG installed to run this example')
} else {

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

##-- Or, read a folder/directory of existing PDB files
#pdb.path <- "my_dir_of_pdbs"
#files <- list.files(path=pdb.path ,
#                   pattern=".pdb",
#                   full.names=TRUE)

##-- Align these PDB sequences
aln <- mustang(files)

##-- Read Aligned PDBs storing coordinate data
pdbs <- read.fasta.pdb(aln)

}

## End(Not run)
```

---

network.amendment	<i>Amendment of a CNA Network According To A Input Community Membership Vector.</i>
-------------------	---

---

**Description**

This function changes the ‘communities’ attribute of a ‘cna’ class object to match a given membership vector.

**Usage**

```
network.amendment(x, membership, minus.log=TRUE)
```

**Arguments**

x	A protein network graph object as obtained from the ‘cna’ function.
membership	A numeric vector containing the new community membership.
minus.log	Logical. Whether to use the minus.log on the cij values.

## Details

This function is useful, in combination with ‘community.tree’, for inspecting different community partitioning options of a input ‘cna’ object. See examples.

## Value

Returns a ‘cna’ class object with the attributes changed according to the membership vector provided.

## Author(s)

Guido Scarabelli

## See Also

[cna](#), [community.tree](#), [summary.cna](#)

## Examples

```
# PDB server connection required - testing excluded

if (!requireNamespace("igraph", quietly = TRUE)) {
  message('Need igraph installed to run this example')
} else {

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

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

## Produce a new k=7 membership vector and CNA network
memb.k7 <- tree$tree[ tree$num.of.comms == 7, ]
net.7 <- network.amendment(net, memb.k7)

plot(net.7, pdb)

print(net)
print(net.7)

}
```



---

`nma`*Normal Mode Analysis*

---

**Description**

Perform normal mode analysis (NMA) on either a single or an ensemble of protein structures.

**Usage**

```
nma(...)
```

**Arguments**

```
...
```

arguments passed to the methods `nma.pdb`, or `nma.pdbs`.

For function `nma.pdb` this will include an object of class `pdb` as obtained from function `read.pdb`.

For function `nma.pdbs` an object of class `pdbs` as obtained from function `pdbaln` or `read.fasta.pdb`.

**Details**

Normal mode analysis (NMA) is a computational approach for studying and characterizing protein flexibility. Current functionality entails normal modes calculation on either a single protein structure or an ensemble of aligned protein structures.

This generic `nma` function calls the corresponding methods for the actual calculation, which is determined by the class of the input argument:

Function `nma.pdb` will be used when the input argument is of class `pdb`. The function calculates the normal modes of a C-alpha model of a protein structure.

Function `nma.pdbs` will be used when the input argument is of class `pdbs`. The function will perform normal mode analysis of each PDB structure stored in the `pdbs` object ('ensemble NMA').

See documentation and examples for each corresponding function for more details.

**Author(s)**

Lars Skjaerven

**References**

Skjaerven, L. et al. (2014) *BMC Bioinformatics* **15**, 399. Grant, B.J. et al. (2006) *Bioinformatics* **22**, 2695–2696.

**See Also**

`nma.pdb`, `nma.pdbs`, `pca`.

## Examples

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

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

## Print modes
print(modes)

## Plot modes
plot(modes)

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

## Needs MUSCLE installed - testing excluded

##- Ensemble NMA
if(check.utility("muscle")) {

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

## Sequence Alignment
pdbs <- pdbaln(files, outfile = tempfile())

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

## Plot fluctuation data
plot(modes, pdbs=pdbs)

}
```

---

nma.pdb

*Normal Mode Analysis*

---

## Description

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

**Usage**

```
## S3 method for class 'pdb'
nma(pdb, inds = NULL, ff = 'calpha', pfc.fun = NULL,
    mass = TRUE, temp = 300.0, keep = NULL, hessian = NULL,
    outmodes = NULL, ... )

build.hessian(xyz, pfc.fun, fc.weights = NULL, pdb = NULL, ...)

## S3 method for class 'nma'
print(x, nmodes=6, ...)
```

**Arguments**

pdb	an object of class <code>pdb</code> as obtained from function <a href="#">read.pdb</a> .
inds	atom and xyz coordinate indices obtained from <a href="#">atom.select</a> that selects the elements of <code>pdb</code> upon which the calculation should be based. If not provided the function will attempt to select the calpha atoms automatically (based on function <a href="#">atom.select</a> ).
ff	character string specifying the force field to use: 'calpha', 'anm', 'pfanm', 'reach', or 'sdenm'.
pfc.fun	customized pair force constant ('pfc') function. The provided function should take a vector of distances as an argument to return a vector of force constants. If provided, 'pfc.fun' will override argument <code>ff</code> . See examples below.
mass	logical, if TRUE the Hessian will be mass-weighted.
temp	numerical, temperature for which the amplitudes for scaling the atomic displacement vectors are calculated. Set 'temp=NULL' to avoid scaling.
keep	numerical, final number of modes to be stored. Note that all subsequent analyses are limited to this subset of modes. This option is useful for very large structures and cases where memory may be limiting.
hessian	hessian matrix as obtained from <a href="#">build.hessian</a> . For internal purposes and generally not intended for public use.
outmodes	atom indices as obtained from <a href="#">atom.select</a> ) specifying the atoms to include in the resulting mode object.
xyz	a numeric vector of Cartesian coordinates.
fc.weights	a numeric matrix of size NxN (where N is the number of calpha atoms) containing scaling factors for the pairwise force constants. See examples below.
x	an <code>nma</code> object obtained from <a href="#">nma.pdb</a> .
nmodes	numeric, number of modes to be printed.
...	additional arguments to <a href="#">build.hessian</a> , <a href="#">aa2mass</a> , <code>pfc.fun</code> , and <a href="#">print</a> . One useful option here for dealing with unconventional residues is 'mass.custom', see the <a href="#">aa2mass</a> function for details.

## Details

This function calculates the normal modes of a C-alpha model of a protein structure. A number of force fields are implemented all of which employ the elastic network model (ENM).

The 'calpha' force field - originally developed by Konrad Hinsen - is the recommended one for most applications. It employs a spring force constant differentiating between nearest-neighbour pairs along the backbone and all other pairs. The force constant function was parameterized by fitting to a local minimum of a crambin model using the AMBER94 force field.

See [load.enmff](#) for details of the different force fields.

By default [nma.pdb](#) will diagonalize the mass-weighted Hessian matrix. The resulting mode vectors are moreover scaled by the thermal fluctuation amplitudes.

The implementation under default arguments reproduces the calculation of normal modes (VibrationalModes) in the Molecular Modeling Toolkit (MMTK) package. To reproduce ANM modes set `ff='anm'`, `mass=False`, and `temp=None`.

## Value

Returns an object of class 'nma' with the following components:

modes	numeric matrix with columns containing the normal mode vectors. Mode vectors are converted to unweighted Cartesian coordinates when <code>mass=True</code> . Note that the 6 first trivial eigenvectors appear in columns one to six.
frequencies	numeric vector containing the vibrational frequencies corresponding to each mode (for <code>mass=True</code> ).
force.constants	numeric vector containing the force constants corresponding to each mode (for <code>mass=False</code> ).
fluctuations	numeric vector of atomic fluctuations.
U	numeric matrix with columns containing the raw eigenvectors. Equals to the modes component when <code>mass=False</code> and <code>temp=None</code> .
L	numeric vector containing the raw eigenvalues.
xyz	numeric matrix of class xyz containing the Cartesian coordinates in which the calculation was performed.
mass	numeric vector containing the residue masses used for the mass-weighting.
temp	numerical, temperature for which the amplitudes for scaling the atomic displacement vectors are calculated.
triv.modes	number of trivial modes.
natoms	number of C-alpha atoms.
call	the matched call.

## Note

The current version provides an efficient implementation of NMA with execution time comparable to similar software (when the entire Hessian is diagonalized).

The main (speed related) bottleneck is currently the diagonalization of the Hessian matrix which is performed with the core R function `eigen`. For computing a few (5-20) approximate modes the user can consult package `'irlba'`.

NMA is memory extensive and users should be cautious when running larger proteins (>3000 residues). Use `'keep'` to reduce the amount of memory needed to store the final `'nma'` object (the full 3Nx3N Hessian matrix still needs to be allocated).

We thank Edvin Fuglebakk for valuable discussions on the implementation as well as for contributing with testing.

### Author(s)

Lars Skjaerven

### References

Skjaerven, L. et al. (2014) *BMC Bioinformatics* **15**, 399. Grant, B.J. et al. (2006) *Bioinformatics* **22**, 2695–2696. Hinsen, K. et al. (2000) *Chemical Physics* **261**, 25–37.

### See Also

[fluct.nma](#), [mktrj.nma](#), [dccm.nma](#), [overlap](#), [rmsip](#), [load.enmff](#).

### Examples

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

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

## Print modes
print(modes)

## Plot modes
plot(modes)

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

## Not run:
## Use Anisotropic Network Model
modes <- nma(pdb, ff="anm", mass=FALSE, temp=NULL, cutoff=15)

## Use SSE information and SS-bonds
sse <- dssp(pdb, resno=FALSE, full=TRUE)
ss.bonds <- matrix(c(76,94, 64,80, 30,115, 6,127),
                  ncol=2, byrow=TRUE)

## User defined energy function
## Note: Must take a vector of distances
```

```

"my.ff" <- function(r) {
  ifelse( r>15, 0, 1 )
}

## Modes with a user defined energy function
modes <- nma(pdb, pfc.fun=my.ff)

## A more manual approach
sele <- atom.select(pdb, chain='A', eley='CA')
xyz <- pdb$xyz[sele$xyz]

hessian <- build.hessian(xyz, my.ff)
modes <- eigen(hessian)

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

## nma(pdb)
#modes <- nma(pdb, mass.custom=list(CSX=121.166))

## End(Not run)

```

---

nma.pdbs

*Ensemble Normal Mode Analysis*


---

## Description

Perform normal mode analysis (NMA) on an ensemble of aligned protein structures.

## Usage

```

## S3 method for class 'pdbs'
nma(pdbs, fit = TRUE, full = FALSE, subspace = NULL,
    rm.gaps = TRUE, varweight=FALSE,
    outpath = NULL, ncore = 1, progress = NULL, ...)

## S3 method for class 'enma'
print(x, ...)

```

## Arguments

pdbs	a numeric matrix of aligned C-alpha xyz Cartesian coordinates. For example an alignment data structure obtained with <a href="#">read.fasta.pdb</a> or <a href="#">pdbaln</a> .
fit	logical, if TRUE coordinate superposition is performed prior to normal mode calculations.
full	logical, if TRUE return the complete, full structure, 'nma' objects.
subspace	number of eigenvectors to store for further analysis.

rm.gaps	logical, if TRUE obtain the hessian matrices for only atoms in the aligned positions (non-gap positions in all aligned structures). Thus, gap positions are removed from output.
varweight	logical, if TRUE perform weighing of the pair force constants. Alternatively, provide a NxN matrix containing the weights. See function <a href="#">var.xyz</a> .
outpath	character string specifying the output directory to which the PDB structures should be written.
ncore	number of CPU cores used to do the calculation. ncore>1 requires package 'parallel' installed.
x	an enma object obtained from <a href="#">nma.pdbs</a> .
progress	progress bar for use with shiny web app.
...	additional arguments to <a href="#">nma</a> , <a href="#">aa2mass</a> , and <a href="#">print</a> .

### Details

This function performs normal mode analysis (NMA) on a set of aligned protein structures obtained with function [read.fasta.pdb](#) or [pdbaln](#). The main purpose is to provide aligned atomic fluctuations and mode vectors in an automated fashion.

The normal modes are calculated on the full structures as provided by object 'pdbs'. With the input argument 'full=TRUE' the full 'nma' objects are returned together with output 'U.subs' providing the aligned mode vectors. When 'rm.gaps=TRUE' the unaligned atoms are omitted from output. With default arguments 'rmsip' provides RMSIP values for all pairwise structures.

See examples for more details.

### Value

Returns an 'enma' object with the following components:

fluctuations	a numeric matrix containing aligned atomic fluctuations with one row per input structure.
rmsip	a numeric matrix of pair wise RMSIP values (only the ten lowest frequency modes are included in the calculation).
U.subspace	a three-dimensional array with aligned eigenvectors (corresponding to the subspace defined by the first N non-trivial eigenvectors ('U') of the 'nma' object).
L	numeric matrix containing the raw eigenvalues with one row per input structure.
xyz	an object of class 'xyz' containing the Cartesian coordinates in which the calculation was performed. Coordinates are superimposed to the first structure of the pdbs object when 'fit=TRUE'.
full.nma	a list with a nma object for each input structure.

### Author(s)

Lars Skjaerven

## References

Skjaerven, L. et al. (2014) *BMC Bioinformatics* **15**, 399. Grant, B.J. et al. (2006) *Bioinformatics* **22**, 2695–2696.

## See Also

For normal mode analysis on single structure PDB: [nma.pdb](#)

For the analysis of the resulting 'eNMA' object: [mktrj.enma](#), [dccm.enma](#), [plot.enma](#), [cov.enma](#).

Similarity measures: [sip](#), [covoverlap](#), [bhattacharyya](#), [rmsip](#).

Related functionality: [pdbaln](#), [read.fasta.pdb](#).

## Examples

```
# Needs MUSCLE installed - testing excluded

if(check.utility("muscle")) {

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

## Sequence Alignment
pdbs <- pdbaln(files, outfile = tempfile())

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

## Plot fluctuation data
plot(modes, pdbs=pdbs)

## Cluster on Fluctuation similarity
sip <- sip(modes)
hc <- hclust(dist(sip))
col <- cutree(hc, k=3)

## Plot fluctuation data
plot(modes, pdbs=pdbs, col=col)

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

## RMSIP is pre-calculated
heatmap(1-modes$rmsip)

## Bhattacharyya coefficient
bc <- bhattacharyya(modes)
heatmap(1-bc)

}
```



---

normalize.vector	<i>Mass-Weighted Normalized Vector</i>
------------------	--

---

### Description

Normalizes a vector (mass-weighted if requested).

### Usage

```
normalize.vector(x, mass=NULL)
```

### Arguments

x	a numeric vector or matrix to be normalized.
mass	a numeric vector containing the atomic masses for weighting.

### Details

This function normalizes a vector, or alternatively, the column-wise vector elements of a matrix. If atomic masses are provided the vector is mass-weighted.

See examples for more details.

### Value

Returns the normalized vector(s).

### Author(s)

Lars Skjaerven

### References

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

### See Also

[nma](#), [inner.prod](#)

### Examples

```
x <- 1:3
y <- matrix(1:9, ncol = 3, nrow = 3)

normalize.vector(x)
normalize.vector(y)

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

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

## Returns a vector
nv <- normalize.vector(modes$modes[,7])

## Returns a matrix
nv <- normalize.vector(modes$modes[,7:10])

## Mass-weighted
nv <- normalize.vector(modes$modes[,7], mass=modes$mass)
```

---

orient.pdb

*Orient a PDB Structure*

---

### Description

Center, to the coordinate origin, and orient, by principal axes, the coordinates of a given PDB structure or xyz vector.

### Usage

```
orient.pdb(pdb, atom.subset = NULL, verbose = TRUE)
```

### Arguments

pdb	a pdb data structure obtained from <a href="#">read.pdb</a> 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 server connection required - testing excluded

pdb <- read.pdb( "1bg2" )
xyz <- orient.pdb(pdb)
#write.pdb(pdb, xyz = xyz, file = "mov1.pdb")

# Based on C-alphas
inds <- atom.select(pdb, "calpha")
xyz <- orient.pdb(pdb, atom.subset=inds$atom)
#write.pdb(pdb, xyz = xyz, file = "mov2.pdb")

# Based on a central Beta-strand
inds <- atom.select(pdb, resno=c(224:232), eley='CA')
xyz <- orient.pdb(pdb, atom.subset=inds$atom)
#write.pdb(pdb, xyz = xyz, file = "mov3.pdb")
```

---

overlap

*Overlap analysis*

---

**Description**

Calculate the squared overlap between sets of vectors.

**Usage**

```
overlap(modes, dv, nmodes=20)
```

**Arguments**

modes	an object of class "pca" or "nma" as obtained from function <code>pca.xyz</code> or <code>nma</code> . Alternatively a 3NxM matrix of eigenvectors can be provided.
dv	a displacement vector of length 3N.
nmodes	the number of modes in which the calculation should be based.

**Details**

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

By definition the cumulative sum of the overlap values equals to one.

Structure modes\$U (or alternatively, the 3NxM matrix of eigenvectors) should be of same length (3N) as dv.

**Value**

Returns a list with the following components:

overlap	a numeric vector of the squared dot products (overlap values) between the (normalized) vector (dv) and each mode in mode.
overlap.cum	a numeric vector of the cumulative squared overlap values.

**Author(s)**

Lars Skjaerven

**References**

Skjaerven, L. et al. (2011) *Proteins* **79**, 232–243. Grant, B.J. et al. (2006) *Bioinformatics* **22**, 2695–2696.

**See Also**

[rmsip](#), [pca.xyz](#), [nma](#), [difference.vector](#)

**Examples**

```
attach(kinesin)

# Ignore gap containing positions
##gaps.res <- gap.inspect(pdb$ali)
gaps.pos <- gap.inspect(pdb$xyz)

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

# Define a difference vector between two structural states
diff.inds <- c(grep("d1v8ka", pdb$id),
              grep("d1goja", pdb$id))

dv <- difference.vector( pdb$xyz[diff.inds,], gaps.pos$f.inds )

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

```

# Plot results
plot(o$overlap, type='h', ylim=c(0,1))
points(o$overlap)
lines(o$overlap.cum, type='b', col='red')

detach(kinesin)

## Not run:
## Calculate overlap from NMA
pdb.a <- read.pdb("1cmk")
pdb.b <- read.pdb("3dnd")

## Fetch CA coordinates
sele.a <- atom.select(pdb.a, chain='E', resno=c(15:350), eley='CA')
sele.b <- atom.select(pdb.b, chain='A', resno=c(1:350), eley='CA')

xyz <- rbind(pdb.a$xyz[sele.a$xyz],
            pdb.b$xyz[sele.b$xyz])

## Superimpose
xyz[2,] <- fit.xyz(xyz[1,], xyz[2,], 1:ncol(xyz))

## The difference between the two conformations
dv <- difference.vector( xyz )

## Calculate normal modes
modes <- nma(pdb.a, inds=sele.a)

# Calculate the squared overlap between the normal modes
# and the difference vector
o <- overlap(modes, dv)

## End(Not run)

```

---

pairwise

*Pair Indices*


---

### Description

A utility function to determine indices for pairwise comparisons.

### Usage

```
pairwise(N)
```

### Arguments

**N** a single numeric value representing the total number of things to undergo pairwise comparison.

**Value**

Returns a two column numeric matrix giving the indices for all pairs.

**Author(s)**

Barry Grant

**References**

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

**See Also**

[seqidentity](#)

**Examples**

```
pairwise(3)
pairwise(20)
```

---

pca

*Principal Component Analysis*

---

**Description**

Performs principal components analysis (PCA) on biomolecular structure data.

**Usage**

```
pca(...)
```

**Arguments**

... arguments passed to the methods `pca.xyz`, `pca.pdbs`, etc. Typically this includes either a numeric matrix of Cartesian coordinates with a row per structure/frame (function `pca.xyz()`), or an object of class `pdbs` as obtained from function `pdbaln` or `read.fasta.pdb` (function `pca.pdbs()`).

**Details**

Principal component analysis can be performed on any structure dataset of equal or unequal sequence composition to capture and characterize inter-conformer relationships.

This generic `pca` function calls the corresponding methods function for actual calculation, which is determined by the class of the input argument `x`. Use `methods("pca")` to list all the current methods for `pca` generic. These will include:

[pca.xyz](#), which will be used when `x` is a numeric matrix containing Cartesian coordinates (e.g. trajectory data).

[pca.pdbs](#), which will perform PCA on the Cartesian coordinates of a input pdbs object (as obtained from the 'read.fasta.pdb' or 'pdbaln' functions).

Currently, function [pca.tor](#) should be called explicitly as there are currently no defined 'tor' object classes.

See the documentation and examples for each individual function for more details and worked examples.

### Author(s)

Barry Grant, Lars Skjaerven

### References

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

### See Also

[pca.xyz](#), [pca.pdbs](#), [pdbaln](#).

---

pca.array

*Principal Component Analysis of an array of matrices*

---

### Description

Calculate the principal components of an array of correlation or covariance matrices.

### Usage

```
## S3 method for class 'array'
pca(x, use.svd = TRUE, rm.gaps=TRUE, ...)
```

### Arguments

x	an array of matrices, e.g. correlation or covariance matrices as obtained from functions dccm or enma2covs.
use.svd	logical, if TRUE singular value decomposition (SVD) is called instead of eigenvalue decomposition.
rm.gaps	logical, if TRUE gap cells (with missing coordinate data in any input matrix) are removed before calculation. This is equivalent to removing NA cells from x.
...	.

### Details

This function performs PCA of symmetric matrices, such as distance matrices from an ensemble of crystallographic structures, residue-residue cross-correlations or covariance matrices derived from ensemble NMA or MD simulation replicates, and so on. The 'upper triangular' region of the matrix is regarded as a long vector of random variables. The function returns  $M$  eigenvalues and eigenvectors with each eigenvector having the dimension  $N(N-1)/2$ , where  $M$  is the number of matrices and  $N$  the number of rows/columns of matrices.

**Value**

Returns a list with components equivalent to the output from `pca.xyz`.

**Author(s)**

Xin-Qiu Yao, Lars Skjaerven

**References**

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

**See Also**

[pca.xyz](#)

---

pca.pdbs

*Principal Component Analysis*

---

**Description**

Performs principal components analysis (PCA) on an ensemble of PDB structures.

**Usage**

```
## S3 method for class 'pdbs'  
pca(pdbs, core.find = FALSE, fit = FALSE, ...)
```

**Arguments**

<code>pdbs</code>	an object of class <code>pdbs</code> as obtained from function <code>pdbsaln</code> or <code>read.fasta.pdb</code> .
<code>core.find</code>	logical, if TRUE <code>core.find()</code> function will be called to find core positions and coordinates of PDB structures will be fitted based on cores.
<code>fit</code>	logical, if TRUE coordinates of PDB structures will be fitted based on all CA atoms.
<code>...</code>	additional arguments passed to the method <code>pca.xyz</code> .

**Details**

The function `pca.pdbs` is a wrapper for the function [pca.xyz](#), wherein more details of the PCA procedure are documented.



**Value**

Returns a list with the following components:

L	eigenvalues.
U	eigenvectors (i.e. the variable loadings).
z.u	scores of the supplied data on the pcs.
sdev	the standard deviations of the pcs.
mean	the means that were subtracted.

**Author(s)**

Barry Grant, Lars Skjaerven and Xin-Qiu Yao

**References**

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

**See Also**

[pca](#), [pca.xyz](#), [pdbaln](#), [nma](#).

**Examples**

```
attach(transducin)

#-- Do PCA ignoring gap containing positions
pc.xray <- pca(pdb)

# Plot results (conformer plots & scree plot)
plot(pc.xray, col=annotation[, "color"])

detach(transducin)
```

---

pca.tor

*Principal Component Analysis*

---

**Description**

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

**Usage**

```
## S3 method for class 'tor'
pca(data, ...)
```

## Arguments

data            numeric matrix of torsion angles with a row per structure.  
...            additional arguments passed to the method `pca.xyz`.

## Value

Returns a list with the following components:

L            eigenvalues.  
U            eigenvectors (i.e. the variable loadings).  
z.u          scores of the supplied data on the pcs.  
sdev        the standard deviations of the pcs.  
mean        the means that were subtracted.

## Author(s)

Barry Grant and Karim ElSawy

## References

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

## See Also

[torsion.xyz](#), [plot.pca](#), [plot.pca.loadings](#), [pca.xyz](#)

## Examples

```
##-- PCA on torsion data for multiple PDBs
attach(kinesin)

gaps.pos <- gap.inspect(pdb$xyz)
tor <- t(apply( pdb$xyz[, gaps.pos$f.inds], 1, torsion.xyz, atm.inc=1))
pc.tor <- pca.tor(tor[, -c(1,233,234,235)])
#plot(pc.tor)
plot.pca.loadings(pc.tor)

detach(kinesin)

## Not run:
##-- PCA on torsion data from an MD trajectory
trj <- read.dcd( system.file("examples/hivp.dcd", package="bio3d") )
tor <- t(apply(trj, 1, torsion.xyz, atm.inc=1))
gaps <- gap.inspect(tor)
pc.tor <- pca.tor(tor[, gaps$f.inds])
plot.pca.loadings(pc.tor)

## End(Not run)
```

---

pca.xyz *Principal Component Analysis*

---

**Description**

Performs principal components analysis (PCA) on a xyz numeric data matrix.

**Usage**

```
## S3 method for class 'xyz'
pca(xyz, subset = rep(TRUE, nrow(as.matrix(xyz))),
     use.svd = FALSE, rm.gaps=FALSE, mass = NULL, ...)

## S3 method for class 'pca'
print(x, nmodes=6, ...)
```

**Arguments**

xyz	numeric matrix of Cartesian coordinates with a row per structure.
subset	an optional vector of numeric indices that selects a subset of rows (e.g. experimental structures vs molecular dynamics trajectory structures) from the full xyz matrix. Note: the full xyz is projected onto this subspace.
use.svd	logical, if TRUE singular value decomposition (SVD) is called instead of eigenvalue decomposition.
rm.gaps	logical, if TRUE gap positions (with missing coordinate data in any input structure) are removed before calculation. This is equivalent to removing NA cols from xyz.
x	an object of class pca, as obtained from function pca.xyz.
nmodes	numeric, number of modes to be printed.
mass	a 'pdb' object or numeric vector of residue/atom masses. By default (mass=NULL), mass is ignored. If provided with a 'pdb' object, masses of all amino acids obtained from <a href="#">aa2mass</a> are used.
...	additional arguments to <a href="#">fit.xyz</a> (for pca.xyz) or to print (for print.pca).

**Value**

Returns a list with the following components:

L	eigenvalues.
U	eigenvectors (i.e. the x, y, and z variable loadings).
z	scores of the supplied xyz on the pcs.
au	atom-wise loadings (i.e. xyz normalized eigenvectors).
sdev	the standard deviations of the pcs.
mean	the means that were subtracted.

**Note**

If mass is provided, mass weighted coordinates will be considered, and iteration of fitting onto the mean structure is performed internally. The extra fitting process is to remove external translation and rotation of the whole system. With this option, a direct comparison can be made between PCs from [pca.xyz](#) and vibrational modes from [nma.pdb](#), with the fact that

$$A = k_B T F^{-1}$$

, where  $A$  is the variance-covariance matrix,  $F$  the Hessian matrix,  $k_B$  the Boltzmann's constant, and  $T$  the temperature.

**Author(s)**

Barry Grant

**References**

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

**See Also**

[pca](#), [pca.pdbs](#), [plot.pca](#), [mktrj.pca](#), [pca.tor](#), [project.pca](#)

**Examples**

```
## Not run:
##-- Read transducin alignment and structures
aln <- read.fasta(system.file("examples/transducin.fa",package="bio3d"))
pdbs <- read.fasta.pdb(aln)

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

rm(list=c("pdbs", "core"))

## End(Not run)

##-- OR for demo purposes just read previously saved transducin data
attach(transducin)

# Previously fitted coordinates based on sub 1.0A^3 core. See core.find() function.
xyz <- pdbs$xyz

##-- Do PCA ignoring gap containing positions
pc.xray <- pca.xyz(xyz, rm.gaps=TRUE)

# Plot results (conformer plots & scree plot overview)
plot(pc.xray, col=annotation[, "color"])
```

```

# Plot a single conformer plot of PC1 v PC2
plot(pc.xray, pc.axes=1:2, col=annotation[, "color"])

## Plot atom wise loadings
plot.bio3d(pc.xray$au[,1], ylab="PC1 (A)")

# PDB server connection required - testing excluded

## Plot loadings in relation to reference structure 1TAG
pdb <- read.pdb("1tag")
ind <- grep("1TAG", pdbc$id)          ## location in alignment

resno <- pdbc$resno[ind, !is.gap(pdbc)] ## non-gap residues
tpdb <- trim.pdb(pdb, resno=resno)

op <- par(no.readonly=TRUE)
par(mfrow = c(3, 1), cex = 0.6, mar = c(3, 4, 1, 1))
plot.bio3d(pc.xray$au[,1], resno, ylab="PC1 (A)", sse=tpdb)
plot.bio3d(pc.xray$au[,2], resno, ylab="PC2 (A)", sse=tpdb)
plot.bio3d(pc.xray$au[,3], resno, ylab="PC3 (A)", sse=tpdb)
par(op)

## Not run:
# Write PC trajectory
resno = pdbc$resno[1, !is.gap(pdbc)]
resid = aa123(pdbc$ali[1, !is.gap(pdbc)])

a <- mktrj.pca(pc.xray, pc=1, file="pc1.pdb",
              resno=resno, resid=resid )

b <- mktrj.pca(pc.xray, pc=2, file="pc2.pdb",
              resno=resno, resid=resid )

c <- mktrj.pca(pc.xray, pc=3, file="pc3.pdb",
              resno=resno, resid=resid )

## End(Not run)

detach(transducin)

```

### Description

Get customizable annotations for query results from PDB or PFAM.

**Usage**

```

pdb.annotate(ids, anno.terms = NULL, unique = FALSE, verbose = FALSE,
             extra.terms = NULL)
pdb.pfam(ids, best.only = TRUE, compact = TRUE)

```

**Arguments**

ids	A character vector of one or more 4-letter PDB codes/identifiers of the files for query, or a 'blast' object containing 'pdb.id'.
anno.terms	Terms can be used for query. The "anno.terms" can be "structureId", "chainId", "macromoleculeType", "chainLength", "experimentalTechnique", "resolution", "scopDomain", "pfam", "ligandId", "ligandName", "source", "structureTitle", "citation", "rObserved", "rFree", "rWork", and "spaceGroup". If anno.terms=NULL, all information would be returned.
unique	logical, if TRUE only unique PDB entries are returned. Alternatively data for each chain ID is provided.
verbose	logical, if TRUE more details are printed.
extra.terms	Additional annotation terms to retrieve from PDB. Currently not supported.
best.only	logical, if TRUE only the lowest eValue match for a given input id will be reported. Otherwise all significant matches will be returned.
compact	logical, if TRUE only a subset of annotation terms are returned. Otherwise full match details are reported (see examples).

**Details**

Given a list of PDB IDs (and query terms for the `pdb.annotate` function), these functions will download annotation information from the RCSB PDB and PFAM databases.

**Value**

Returns a data frame of query results with a row for each PDB record, and annotation terms column-wise.

**Author(s)**

Hongyang Li, Barry Grant, Lars Skjaerven, Xin-Qiu Yao

**Examples**

```

# PDB server connection required - testing excluded

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

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

```

```
## only unique PDB IDs
anno <- pdb.annotate(ids, unique=TRUE)

# Fetch only specific terms
pdb.annotate(ids, anno.terms = c("pfam", "ligandId", "citation"))

## Not run:
# PFAM server connection required - testing excluded

# Find PFAM annotations of PDB entries
pdb.pfam(c("6Q21_A", "1NVW", "1P2U_A"))

# More details and a not found entry warning
pdb.pfam(c("1P2U_A", "6Q21_B"), compact=FALSE)

## End(Not run)
```

---

pdb2aln

*Align a PDB structure to an existing alignment*

---

## Description

Extract sequence from a PDB object and align it to an existing multiple sequence alignment that you wish keep intact.

## Usage

```
pdb2aln(aln, pdb, id="seq.pdb", aln.id=NULL, file="pdb2aln.fa", ...)
```

## Arguments

aln	an alignment list object with id and ali components, similar to that generated by <a href="#">read.fasta</a> , <a href="#">read.fasta.pdb</a> , and <a href="#">seqaln</a> .
pdb	the PDB object to be added to aln.
id	name for the PDB sequence in the generated new alignment.
aln.id	id of the sequence in aln that is close to the sequence from pdb.
file	output file name for writing the generated new alignment.
...	additional arguments passed to <a href="#">seqaln</a> .

## Details

The basic effect of this function is to add a PDB sequence to an existing alignment. In this case, the function is simply a wrapper of [seq2aln](#).

The more advanced (and also more useful) effect is giving complete mappings from the column indices of the original alignment (`aln$ali`) to atomic indices of equivalent C-alpha atoms in the pdb. These mappings are stored in the output list (see below 'Value' section). This feature is better illustrated in the function [pdb2aln.ind](#), which calls `pdb2aln` and directly returns atom selections given a set of alignment positions. (See [pdb2aln.ind](#) for details. )

When `aln.id` is provided, the function will do pairwise alignment between the sequence from `pdb` and the sequence in `aln` with `id` matching `aln.id`. This is the best way to use the function if the protein has an identical or very similar sequence to one of the sequences in `aln`.

## Value

Return a list object of the class 'fasta' containing three components:

<code>id</code>	sequence names as identifiers.
<code>ali</code>	an alignment character matrix with a row per sequence and a column per equivalent aminoacid/nucleotide.
<code>ref</code>	an integer 2xN matrix, where N is the number of columns of the new alignment <code>ali</code> . The first row contains the column indices of the original alignment <code>aln\$ali</code> . The second row contains atomic indices of equivalent C-alpha atoms in <code>pdb</code> . Gaps in the new alignment are indicated by NAs.

## Author(s)

Xin-Qiu Yao & Barry Grant

## References

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

## See Also

[seqaln](#), [seq2aln](#), [seqaln.pair](#), [pdb2aln.ind](#)

## Examples

```
## Not run:
##--- Read aligned PDB coordinates (CA only)
aln <- read.fasta(system.file("examples/kif1a.fa",package="bio3d"))
pdbs <- read.fasta.pdb(aln)

##--- Read PDB coordinate for a new structure (all atoms)
id <- get.pdb("2kin", URLonly=TRUE)
pdb <- read.pdb(id)

# add pdb to the alignment
naln <- pdb2aln(aln=pdbs, pdb=pdb, id=id)
```



```
aln

## End(Not run)
```

---

pdb2aln.ind

*Mapping from alignment positions to PDB atomic indices*

---

### Description

Find the best alignment between a PDB structure and an existing alignment. Then, given a set of column indices of the original alignment, returns atom selections of equivalent C-alpha atoms in the PDB structure.

### Usage

```
pdb2aln.ind(aln, pdb, inds = NULL, ...)
```

### Arguments

aln	an alignment list object with <code>id</code> and <code>ali</code> components, similar to that generated by <code>read.fasta</code> , <code>read.fasta.pdb</code> , <code>pdbaln</code> , and <code>seqaln</code> .
pdb	the PDB object to be aligned to <code>aln</code> .
inds	a numeric vector containing a subset of column indices of <code>aln</code> . If <code>NULL</code> , non-gap positions of <code>aln\$ali</code> are used.
...	additional arguments passed to <code>pdb2aln</code> .

### Details

Call `pdb2aln` to align the sequence of `pdb` to `aln`. Then, find the atomic indices of C-alpha atoms in `pdb` that are equivalent to `inds`, the subset of column indices of `aln$ali`.

The function is a routine utility in a combined analysis of molecular dynamics (MD) simulation trajectories and crystallographic structures. For example, a typical post-analysis of MD simulation is to compare the principal components (PCs) derived from simulation trajectories with those derived from crystallographic structures. The C-alpha atoms used to fit trajectories and do PCA must be the same (or equivalent) to those used in the analysis of crystallographic structures, e.g. the 'non-gap' alignment positions. Call `pdb2aln.ind` with providing relevant alignment positions, one can easily get equivalent atom selections ('select' class objects) for the simulation topology (PDB) file and then do proper trajectory analysis.

**Value**

Returns a list containing two "select" objects:

a                   atom and xyz indices for the alignment.  
b                   atom and xyz indices for the PDB.

Note that if any element of `inds` has no corresponding CA atom in the PDB, the output `a$atom` and `b$atom` will be shorter than `inds`, i.e. only indices having equivalent CA atoms are returned.

**Author(s)**

Xin-Qiu Yao, Lars Skjaerven & Barry Grant

**References**

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

**See Also**

[seq2aln](#), [seqaln.pair](#), [pdb2aln](#)

**Examples**

```
## Not run:
##--- Read aligned PDB coordinates (CA only)
aln <- read.fasta(system.file("examples/kif1a.fa",package="bio3d"))
pdbs <- read.fasta.pdb(aln)

##--- Read the topology file of MD simulations
##--- For illustration, here we read another pdb file (all atoms)
pdb <- read.pdb("2kin")

#--- Map the non-gap positions to PDB C-alpha atoms
#pc.inds <- gap.inspect(pdbs$ali)
#npc.inds <- pdb2aln.ind(aln=pdbs, pdb=pdb, inds=pc.inds$f.inds)

#npc.inds$a
#npc.inds$b

#--- Or, map the non-gap positions with a known close sequence in the alignment
#npc.inds <- pdb2aln.ind(aln=pdbs, pdb=pdb, aln.id="1bg2", inds=pc.inds$f.inds)

#--- Map core positions
core <- core.find(pdbs)
core.inds <- pdb2aln.ind(aln=pdbs, pdb=pdb, inds = core$c1A.atom)

core.inds$a
core.inds$b

##--- Fit simulation trajectories to one of the X-ray structures based on
##--- core positions
#xyz <- fit.xyz(pdbs$xyz[1,], pdb$xyz, core.inds$a$xyz, core.inds$b$xyz)
```

```
##--- Do PCA of trajectories based on non-gap positions
#pc.traj <- pca(xyz[, npc.inds$b$xyz])

## End(Not run)
```

---

`pdb2sse`

*Obtain An SSE Sequence Vector From A PDB Object*

---

### Description

Results are similar to that returned by `stride(pdb)$sse` and `dssp(pdb)$sse`.

### Usage

```
pdb2sse(pdb, verbose = TRUE)
```

### Arguments

`pdb` an object of class `pdb` as obtained from function [read.pdb](#).  
`verbose` logical, if TRUE warnings and other messages will be printed.

### Details

call for its effects.

### Value

a character vector indicating SSE elements for each amino acid residue. The 'names' attribute of the vector contains 'resno', 'chain', 'insert', and 'SSE segment number', seperated by the character '\_'.

### Author(s)

Barry Grant & Xin-Qiu Yao

### See Also

[dssp](#), [stride](#), [bounds.sse](#)

## Examples

```
#PDB server connection required - testing excluded

pdb <- read.pdb("1a7l")
sse <- pdb2sse(pdb)
sse
```

---

pdbaln

*Sequence Alignment of PDB Files*

---

## Description

Create multiple sequences alignments from a list of PDB files returning aligned sequence and structure records.

## Usage

```
pdbaln(files, fit = FALSE, pqr = FALSE, ncore = 1,
        nseg.scale = 1, progress = NULL, ...)
```

## Arguments

files	a character vector of PDB file names. Alternatively, a list of pdb objects can be provided.
fit	logical, if TRUE coordinate superposition is performed on the input structures.
pqr	logical, if TRUE the input structures are assumed to be in PQR format.
ncore	number of CPU cores used to do the calculation. <code>ncore&gt;1</code> requires package 'parallel' installed.
nseg.scale	split input data into specified number of segments prior to running multiple core calculation. See <a href="#">fit.xyz</a> .
progress	progress bar for use with shiny web app.
...	extra arguments passed to <code>seqaln</code> function.

## Details

This wrapper function calls the underlying functions `read.pdb`, `pdbseq`, `seqaln` and `read.fasta.pdb` returning a list of class "pdbs" similar to that returned by `read.fasta.pdb`.

As these steps are often error prone it is recommended for most cases that the individual underlying functions are called in sequence with checks made on the validity of their respective outputs to ensure sensible results.

**Value**

Returns a list of class "pdbc" with the following five components:

xyz	numeric matrix of aligned C-alpha coordinates.
resno	character matrix of aligned residue numbers.
b	numeric matrix of aligned B-factor values.
chain	character matrix of aligned chain identifiers.
id	character vector of PDB sequence/structure names.
ali	character matrix of aligned sequences.
call	the matched call.

**Note**

See recommendation in details section above.

**Author(s)**

Barry Grant

**References**

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

**See Also**

[read.pdb](#), [pdbseq](#), [seqaln](#), [read.fasta](#), [read.fasta.pdb](#), [core.find](#), [fit.xyz](#), [read.all](#), [pymol.pdbc](#)

**Examples**

```
## Not run:
##- Align PDBs (from vector of filenames)
#files <- get.pdb(c("4q21", "5p21"), URLonly=TRUE)
files <- get.pdb(c("4q21", "5p21"), path=tempdir(), overwrite=TRUE)
pdbaln(files)

##- Align PDBs (from list of existing PDB objects)
pdblast <- list(read.pdb(files[1]), read.pdb(files[2]))
pdbaln(pdblast)

## End(Not run)
```

---

pdbfit *PDB File Coordinate Superposition*

---

### Description

Protein Databank Bank file coordinate superposition with the Kabsch algorithm.

### Usage

```

pdbfit(...)

## S3 method for class 'pdb'
pdbfit(pdb, inds = NULL, ...)

## S3 method for class 'pdbs'
pdbfit(pdbs, inds = NULL, outpath = NULL, ...)

```

### Arguments

pdb	a multi-model pdb object of class "pdb", as obtained from <code>read.pdb</code> .
pdbs	a list of class "pdbs" containing PDB file data, as obtained from <code>read.fasta.pdb</code> or <code>pdbaln</code> .
inds	a list object with a 'xyz' component with indices that selects the coordinate positions (in terms of x, y and z elements) upon which fitting should be based. This defaults to all equivalent non-gap positions for function <code>pdbfit.pdbs</code> , and to all calpha atoms for function <code>pdbfit.pdb</code> .
outpath	character string specifying the output directory for optional coordinate file output. Note that full files (i.e. all atom files) are written, see below.
...	extra arguments passed to <code>fit.xyz</code> function.

### Details

The function `pdbfit` is a wrapper for the function `fit.xyz`, wherein full details of the superposition procedure are documented.

Input to `pdbfit.pdbs` should be a list object obtained with the function `read.fasta.pdb` or `pdbaln`. See the examples below.

For function `pdbfit.pdb` the input should be a multi-model pdb object with multiple (>1) frames in the 'xyz' component.

The reference frame for superposition (i.e. the fixed structure to which others are superposed) is the first entry in the input "pdbs" object. For finer control use `fit.xyz`.

### Value

Returns moved coordinates.

**Author(s)**

Barry Grant

**References**Grant, B.J. et al. (2006) *Bioinformatics* **22**, 2695–2696.Kabsch *Acta Cryst* (1978) **A34**, 827–828.**See Also**[pdbaln](#), [read.fasta.pdb](#), [fit.xyz](#), [rmsd](#), [read.pdb](#)**Examples**

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

# Superpose again this time outputting all-atom PDBs to disc
#xyz <- pdbfit( pdbs, outpath="fitted" )

## End(Not run)
```

---

pdbs2pdb

*PDBs to PDB Converter*

---

**Description**

Convert a list of PDBs from an "pdbs" object to a list of pdb objects.

**Usage**

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

**Arguments**

pdbs	a list of class "pdbs" containing PDB file data, as obtained from <code>read.fasta.pdb</code> , <code>pdbaln</code> , or <code>read.all</code> .
inds	a vector of indices that selects the PDB structures to convert.
rm.gaps	logical, if TRUE atoms in gap containing columns are removed in the output pdb objects.
all.atom	logical, if TRUE all atom data are converted (the 'pdbs' object must be obtained from <code>read.all</code> or <code>pdbs\$id</code> refers to existing PDB files).
ncore	number of CPU cores used to do the calculation.

### Details

This function will generate a list of pdb objects from a "pdbs" class.  
See examples for more details/

### Value

Returns a list of pdb objects.

### Author(s)

Lars Skjaerven

### References

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

### See Also

[read.pdb](#), [pdbaln](#), [read.fasta.pdb](#).

### Examples

```
## Not run:
## Fetch PDBs
pdb.ids <- c("1YX5_B", "3NOB", "1P3Q_U")
outdir <- paste(tempdir(), "/raw_pdb", sep="")
outdir = "raw_pdb"
raw.files <- get.pdb(pdb.ids, path = outdir)

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

## Align and fit
pdbs <- pdbaln(all.files, fit=TRUE)

## Convert back to PDB objects
all.pdbs <- pdbs2pdb(pdbs)

## Access the first PDB object
## all.pdbs[[1]]

## Return PDB objects consisting of only
## atoms in non-gap positions
all.pdbs <- pdbs2pdb(pdbs, rm.gaps=TRUE)

## End(Not run)
```



---

pdbc2sse                      *SSE annotation for a PDBs Object*

---

### Description

Returns secondary structure element (SSE) annotation ("sse" object) for a structure in the provided "pdbc" object.

### Usage

```
pdbc2sse(pdbc, ind = NULL, rm.gaps = TRUE, resno = TRUE, pdb = FALSE, ...)
```

### Arguments

pdbc	a list of class "pdbc" containing PDB file data, as obtained from read.fasta.pdb or pdbc1n.
ind	numeric index pointing to the PDB in which the SSE should be provided. If ind=NULL, then the consensus SSE is returned.
rm.gaps	logical, if TRUE SSEs spanning gap containing columns are omitted from the output in the resulting sse object.
resno	logical, if TRUE output is in terms of residue numbers rather than residue index (position in sequence).
pdb	logical, if TRUE function dssp will be called on the corresponding pdb object rather than to use pdbc\$sse to obtain the SSE object.
...	arguments passed to function dssp.

### Details

This function provides a "sse" list object containing secondary structure elements (SSE) annotation data for a particular structure in the provided "pdbc" object. Residue numbers are provided relative to the alignment in the "pdbc" object.

When ind=NULL the function will attempt to return the consensus SSE annotation, i.e. where there are SSEs across all structures. This will only work SSE data is found in the "pdbc" object.

See examples for more details.

### Value

Returns a list object of class sse.

### Author(s)

Lars Skjaerven

### References

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

**See Also**

[dssp](#), [pdbaln](#), [read.fasta.pdb](#).

**Examples**

```
## Not run:
attach(transducin)

## calculate RMSF
rf <- rmsf(pdb$xyz)

## Fetch SSE annotation, output in terms of alignment index
sse <- pdb$2sse(pdb, ind=1, rm.gaps=FALSE, resno=FALSE)

## Add SSE annotation to plot
plotb3(rf, sse=sse)

## Calculate RMSF only for non-gap columns
gaps.pos <- gap.inspect(pdb$xyz)
rf <- rmsf(pdb$xyz[, gaps.pos$f.inds])

## With gap columns removed, output in terms of residue number
sse <- pdb$2sse(pdb, ind=1, rm.gaps=TRUE, resno=TRUE)
gaps.res <- gap.inspect(pdb$ali)
plotb3(rf, sse=sse, resno=pdb$resno[1, gaps.res$f.inds])

detach(transducin)

## End(Not run)
```

---

pdbseq

*Extract The Aminoacid Sequence From A PDB Object*

---

**Description**

Return a vector of the one-letter IUPAC or three-letter PDB style aminoacid codes from a given PDB object.

**Usage**

```
pdbseq(pdb, inds = NULL, aa1 = TRUE)
```

**Arguments**

pdb	a PDB structure object obtained from <a href="#">read.pdb</a> .
inds	a list object of ATOM and XYZ indices as obtained from <a href="#">atom.select</a> .
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 examples below and 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:

[https://www.insdc.org/documents/feature\\_table.html#7.4.3](https://www.insdc.org/documents/feature_table.html#7.4.3)

For more information on PDB residue codes see:

<http://ligand-expo.rcsb.org/ld-search.html>

**See Also**

[read.pdb](#), [atom.select](#), [aa321](#), [read.fasta](#)

**Examples**

```
## Not run:
pdb <- read.pdb( "5p21" )
pdbseq(pdb)

#pdbseq(pdb, inds=atom.select(pdb, resno=5:15, elety="CA"), aa1=FALSE)

## End(Not run)
```

---

pdbsplit

*Split a PDB File Into Separate Files, One For Each Chain.*

---

**Description**

Split a Protein Data Bank (PDB) coordinate file into new separate files with one file for each chain.

**Usage**

```
pdbsplit(pdb.files, ids = NULL, path = "split_chain", overwrite=TRUE,
         verbose = FALSE, mk4=FALSE, ncore = 1, progress = NULL, ...)
```

### Arguments

<code>pdb.files</code>	a character vector of PDB file names.
<code>ids</code>	a character vector of PDB and chain identifiers (of the form: 'pdbId_chainId', e.g. '1bg2_A'). Used for filtering chain IDs for output (in the above example only chain A would be produced).
<code>path</code>	output path for chain-split files.
<code>overwrite</code>	logical, if FALSE the PDB structures will not be read and written if split files already exist.
<code>verbose</code>	logical, if TRUE details of the PDB header and chain selections are printed.
<code>mk4</code>	logical, if TRUE output filenames will use only the first four characters of the input filename (see <code>basename.pdb</code> for details).
<code>ncore</code>	number of CPU cores used for the calculation. <code>ncore&gt;1</code> requires package 'parallel' be installed.
<code>progress</code>	progress bar for use with shiny web app.
<code>...</code>	additional arguments to <code>read.pdb</code> . Useful e.g. for parsing multi model PDB files, including ALT records etc. in the output files.

### Details

This function will produce single chain PDB files from multi-chain input files. By default all separate filenames are returned. To return only a subset of select chains the optional input 'ids' can be provided to filter the output (e.g. to fetch only chain C, of a PDB object with additional chains A+B ignored). See examples section for further details.

Note that multi model atom records will only split into individual PDB files if `multi=TRUE`, else they are omitted. See examples.

### Value

Returns a character vector of chain-split file names.

### Author(s)

Barry Grant

### References

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

For a description of PDB format (version3.3) see:

<http://www.wwpdb.org/documentation/format33/v3.3.html>.

### See Also

[read.pdb](#), [atom.select](#), [write.pdb](#), [get.pdb](#).

## Examples

```
## Not run:
## Save separate PDB files for each chain of a local or on-line file
pdbsplit( get.pdb("2KIN", URLonly=TRUE) )

## Split several PDBs by chain ID and multi-model records
raw.files <- get.pdb( c("1YX5", "3NOB") , URLonly=TRUE)
chain.files <- pdbsplit(raw.files, path=tempdir(), multi=TRUE)
basename(chain.files)

## Output only desired pdbID_chainID combinations
## for the last entry (1f9j), fetch all chains
ids <- c("1YX5_A", "3NOB_B", "1F9J")
raw.files <- get.pdb( ids , URLonly=TRUE)
chain.files <- pdbsplit(raw.files, ids, path=tempdir())
basename(chain.files)

## End(Not run)
```

---

pfam

*Download Pfam FASTA Sequence Alignment*

---

## Description

Downloads FASTA sequence alignment from the Pfam database.

## Usage

```
pfam(id, alignment = "seed", verbose = FALSE)
```

## Arguments

id	the Pfam family identifier (e.g. 'Piwi') or accession (e.g. 'PF02171').
alignment	the alignment type. Allowed values are: 'seed', 'ncbi', 'full', 'metagenomics'.
verbose	logical, if TRUE details of the download process is printed.

## Details

This is a basic function to download a multiple sequence alignment for a protein family from the Pfam database.

**Value**

A 'fasta' object with the following components:

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

**Note**

Full more information on the Pfam database:

<http://pfam.xfam.org>

**Author(s)**

Lars Skjaerven

**References**

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

**See Also**

[read.fasta](#), [hmm](#), [get.seq](#), [uniprot](#)

**Examples**

```
## Not run:
# PFAM server connection required - testing excluded

aln <- pfam("piwi")
aln <- pfam("PF02171")

seq <- get.seq("1rx2_A", outfile = tempfile())
hmm <- hmmmer(seq, type="hmmscan", db="pfam")
aln <- pfam(hmm$hit.tbl$acc[1])

# Or much more simply for RCSB PDB entries:
acc <- pdb.pfam("1rx2_A", compact=FALSE)$pfamAcc
aln <- pfam(acc)

## 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**

```
plotb3(x, resno = NULL, rm.gaps = FALSE, type = "h",
       main = "", sub = "",
       xlim = NULL, ylim = NULL, ylim2zero = TRUE,
       xlab = "Residue", ylab = NULL,
       axes = TRUE, ann = par("ann"), col = par("col"),
       sse = NULL, sse.type="classic", sse.min.length=5,
       top = TRUE, bot = TRUE,
       helix.col = "gray20", sheet.col = "gray80",
       sse.border = FALSE, ...)

## S3 method for class 'bio3d'

plot(...)
```

**Arguments**

x	a numeric vector of values to be plotted. Any reasonable way of defining these plot coordinates is acceptable. See the function ‘xy.coords’ for details.
resno	an optional vector with length equal to that of ‘x’ that will be used to annotate the xaxis. This is typically a vector of residue numbers. If NULL residue positions from 1 to the length of ‘x’ will be used. See examples below.
rm.gaps	logical, if TRUE gaps in x, indicated by NA values, will be removed from plot.
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 over-plotted 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 colors 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 color specified.
sse	secondary structure object as returned from <a href="#">dssp</a> , <a href="#">stride</a> or in certain cases <a href="#">read.pdb</a> .
sse.type	single element character vector that determines the type of secondary structure annotation drawn. The following values are possible, 'classic' and 'fancy'. See details and examples below.
sse.min.length	a single numeric value giving the length below which secondary structure elements will not be drawn. This is useful for the exclusion of short helix and strand regions that can often crowd these forms of plots.
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

This function is useful for plotting per-residue numeric vectors for a given protein structure (e.g. results from RMSF, PCA, NMA etc.) along with a schematic representation of major secondary structure elements.

Two forms of secondary structure annotation are available: so called 'classic' and 'fancy'. The former draws marginal rectangles and has been available within Bio3D from version 0.1. The later draws more 'fancy' (and distracting) 3D like helices and arrowed strands.

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**

```
# PDB server connection required - testing excluded

## Plot of B-factor values along with secondary structure from PDB
pdb <- read.pdb( "1bg2" )
bfac <- pdb$atom[pdb$calpha,"b"]
plot.bio3d(bfac, sse=pdb, ylab="B-factor", col="gray")
points(bfac, typ="l")

## Not run:
## Use PDB residue numbers and include short secondary structure elements
plot.bio3d(pdb$atom[pdb$calpha,"b"], sse=pdb, resno=pdb, ylab="B-factor",
  typ="l", lwd=1.5, col="blue", sse.min.length=0)

## Calculate secondary structure using stride() or dssp()
#sse <- stride(pdb)
sse <- dssp(pdb)

## Plot of B-factor values along with calculated secondary structure
plot.bio3d(pdb$atom[pdb$calpha,"b"], sse=sse, ylab="B-factor", typ="l",
  col="blue", lwd=2)

## End(Not run)

# PDB server connection required - testing excluded

## Plot 'aligned' data respecting gap positions
attach(transducin)

pdb = read.pdb("1tnd") ## Reference PDB see: pdbname[1]
pdb = trim.pdb(pdb, inds=atom.select(pdb, chain="A"))

## Plot of B-factor values with gaps
plot.bio3d(pdb$b, resno=pdb, sse=pdb, ylab="B-factor")

## Plot of B-factor values after removing all gaps
```

```

plot.bio3d(pdb$b, rm.gaps=TRUE, resno = pdb, sse=pdb, ylab="B-factor")

detach(transducin)

## Fancy secondary structure elements
##plot.bio3d(pdb$atom[pdb$calpha,"b"], sse=pdb, ssetype="fancy")
## Currently not implemented

```

---

plot.cmap

*Plot Contact Matrix*


---

### Description

Plot a contact matrix with optional secondary structure in the marginal regions.

### Usage

```

## S3 method for class 'cmap'
plot(x, col=2, pch=16, main="Contact map", sub="",
      xlim=NULL, ylim=NULL, xlab = "Residue index", ylab = xlab,
      axes=TRUE, ann=par("ann"), sse=NULL, sse.type="classic",
      sse.min.length=5, bot=TRUE, left=TRUE,
      helix.col="gray20", sheet.col="gray80", sse.border=FALSE,
      add=FALSE, ...)

```

### Arguments

x	a numeric matrix of residue contacts as obtained from function <code>cmap</code> .
col	color code or name, see <code>par</code> .
pch	plotting 'character', i.e., symbol to use. This can either be a single character or an integer code for one of a set of graphics symbols. See <code>points</code> .
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.
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.
sse	secondary structure object as returned from <code>dssp</code> , <code>stride</code> or in certain cases <code>read.pdb</code> .

sse.type	single element character vector that determines the type of secondary structure annotation drawn. The following values are possible, 'classic' and 'fancy'. See details and examples below.
sse.min.length	a single numeric value giving the length below which secondary structure elements will not be drawn. This is useful for the exclusion of short helix and strand regions that can often crowd these forms of plots.
left	logical, if TRUE rectangles for each sse are drawn towards the left 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.
add	logical, specifying if the contact map should be added to an already existing plot. Note that when 'TRUE' only points are plotted (no annotation).
...	other graphical parameters.

### Details

This function is useful for plotting a residue-residue contact data for a given protein structure along with a schematic representation of major secondary structure elements.

Two forms of secondary structure annotation are available: so called 'classic' and 'fancy'. The former draws marginal rectangles and has been available within Bio3D from version 0.1. The later draws more 'fancy' (and distracting) 3D like helices and arrowed strands.

### 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)

Lars Skjaerven, Barry Grant

### References

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

### See Also

[cmap](#), [dm](#), [plot.dmat](#), [plot.default](#), [plot.bio3d](#), [dssp](#), [stride](#)

**Examples**

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

##- Calcualte contact map
cm <- cmap(pdb)

##- Plot contact map
plot.cmap(cm, sse=pdb)

##- Add to plot
plot.cmap(t(cm), col=3, pch=17, add=TRUE)
```

---

plot.cna

*Protein Structure Network Plots in 2D and 3D.*


---

**Description**

Plot a protein dynamic network as obtained from the *cna* function.

**Usage**

```
## S3 method for class 'cna'
plot(x, pdb = NULL, weights=NULL, vertex.size=NULL,
      layout=NULL, col=NULL, full=FALSE, scale=TRUE, color.edge = FALSE,
      interactive=FALSE, ...)
## S3 method for class 'ecna'
plot(x, ...)
```

**Arguments**

<code>x</code>	A protein network graph object (or a list of such objects) as obtained from the 'cna' function.
<code>pdb</code>	A PDB structure object obtained from 'read.pdb'. If supplied this will be used to guide the network plot 'layout', see 'layout.cna' for details.
<code>weights</code>	A numeric vector containing the edge weights for the network.
<code>vertex.size</code>	A numeric vector of node/community sizes. If NULL the size will be taken from the input network graph object 'x'. Typically for 'full=TRUE' nodes will be of an equal size and for 'full=FALSE' community node size will be proportional to the residue membership of each community.
<code>layout</code>	Either a function or a numeric matrix. It specifies how the vertices will be placed on the plot. See 'layout.cna'.
<code>col</code>	A vector of colors used for node/vertex rendering. If NULL these values are taken from the input network 'V(x\$community.network)\$color'.

full	Logical, if TRUE the full all-atom network rather than the clustered community network will be plotted.
scale	Logical, if TRUE weights are scaled with respect to the network.
color.edge	Logical, if TRUE edges are colored with respect to their weights.
interactive	Logical, if TRUE interactive graph will be drawn where users can manually adjust the network (positions of vertices, colors of edges, etc.). Needs Tcl/Tk support in the installed R build.
...	Additional graphical parameters for 'plot.igraph'.

### Details

This function calls 'plot.igraph' from the igraph package to plot cna networks the way we like them.

The plot layout is user settable, we like the options of: 'layout.cna', 'layout.fruchterman.reingold', 'layout.mds' or 'layout.svd'. Note that first of these uses PDB structure information to produce a more meaningful layout.

Extensive plot modifications are possible by setting additional graphical parameters (...). These options are detailed in 'igraph.plotting'. Common parameters to alter include:

**vertex.label:** Node labels,  $V(x\$network)\$name$ . Use NA to omit.

**vertex.label.color:** Node label colors, see also `vertex.label.cex` etc.

**edge.color:** Edge colors,  $E(x\$network)\$color$ .

**mark.groups:** Community highlighting, a community list object, see also `mark.col` etc.

### Value

Produces a network plot on the active graphics device. Also returns the plot layout coordinates silently, which can be passed to the 'identify.cna' function.

### Note

Be sure to check the correspondence of your 'pdb' object with your network object 'x', as few internal checks are currently performed by the 'layout.cna' function.

### Author(s)

Barry Grant and Guido Scarabelli

### References

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

### See Also

[plot.igraph](#), [plot.communities](#), [igraph.plotting](#)

**Examples**

```

# PDB server connection required - testing excluded

if (!requireNamespace("igraph", quietly = TRUE)) {
  message('Need igraph installed to run this example')
} else {

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

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

  # Chose a different PDB informed layout for plot
  plot.cna(net, pdb)

  # Play with plot layout and colors...
  plot.cna(net, layout=igraph::layout.mds(net$community.network), col=c("blue","green") )

  # Plot full residue network colored by communities - will be slow due to number of edges!!
  plot.cna(net, pdb, full=TRUE)

  # Alter plot settings
  plot.cna(net, pdb, full=TRUE, vertex.size=3, weights=1, vertex.label=NA)

}

```

---

plot.core

*Plot Core Fitting Progress*


---

**Description**

Plots the total ellipsoid volume of core positions versus core size at each iteration of the core finding process.

**Usage**

```

## S3 method for class 'core'
plot(x, y = NULL, type = "h", main = "", sub = "",
     xlim = NULL, ylim = NULL, xlab = "Core Size (Number of Residues)",
     ylab = "Total Ellipsoid Volume (Angstrom^3)", axes = TRUE,
     ann = par("ann"), col = par("col"), ...)

```

**Arguments**

x	a list object obtained with the function <code>core.find</code> from which the 'volume' component is taken as the x coordinates for the plot.
y	the y coordinates for the plot.
type	one-character string giving the type of plot desired.
main	a main title for the plot, see also 'title'.
sub	a sub-title for the plot.
xlim	the x limits of the plot.
ylim	the y limits of the plot.
xlab	a label for the x axis.
ylab	a label for the y axis.
axes	a logical value indicating whether both axes should be drawn.
ann	a logical value indicating whether the default annotation (title and x and y axis labels) should appear on the plot.
col	The colors for lines and points. Multiple colours can be specified so that each point is given its own color. If there are fewer colors than points they are recycled in the standard fashion.
...	extra plotting arguments.

**Value**

Called for its effect.

**Note**

The produced plot can be useful for deciding on the core/non-core boundary.

**Author(s)**

Barry Grant

**References**

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

**See Also**

[core.find](#), [print.core](#)

**Examples**

```

## Not run:

##-- Generate a small kinesin alignment and read corresponding structures
pdbfiles <- get.pdb(c("1bg2","2ncd","1i6i","1i5s"), URLonly=TRUE)
pdbs <- pdbaln(pdbfiles)

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

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

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

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

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

## End(Not run)

```

---

plot.dccm

*DCCM Plot*


---

**Description**

Plot a dynamical cross-correlation matrix.

**Usage**

```

## S3 method for class 'dccm'
plot(x, resno=NULL, sse=NULL, colorkey=TRUE,
      at=c(-1, -0.75, -0.5, -0.25, 0.25, 0.5, 0.75, 1),
      main="Residue Cross Correlation",
      helix.col = "gray20", sheet.col = "gray80",
      inner.box=TRUE, outer.box=FALSE,
      xlab="Residue No.", ylab="Residue No.",
      margin.segments=NULL, segment.col=vmd_colors(), segment.min=1, ...)

```



**Arguments**

x	a numeric matrix of atom-wise cross-correlations as output by the ‘dccm’ function.
resno	an optional vector with length equal to that of x that will be used to annotate the x- and y-axis. This is typically a vector of residue numbers. Can be also provided with a ‘pdb’ object, in which ‘resno’ of all C-alpha atoms will be used. If NULL residue positions from 1 to the length of x will be used. See examples below.
sse	secondary structure object as returned from <a href="#">dssp</a> , <a href="#">stride</a> or <a href="#">read.pdb</a> .
colorkey	logical, if TRUE a key is plotted.
at	numeric vector specifying the levels to be colored.
main	a main title for the plot.
helix.col	The colors for rectangles representing alpha helices.
sheet.col	The colors for rectangles representing beta strands.
inner.box	logical, if TRUE an outer box is drawn.
outer.box	logical, if TRUE an outer box is drawn.
xlab	a label for the x axis.
ylab	a label for the y axis.
margin.segments	a numeric vector of cluster membership as obtained from <a href="#">cutree()</a> or other community detection method. This will be used for bottom and left margin annotation.
segment.col	a vector of colors used for each cluster group in <a href="#">margin.segments</a> .
segment.min	a single element numeric vector that will cause <a href="#">margin.segments</a> with a length below this value to be excluded from the plot.
...	additional graphical parameters for <a href="#">contourplot</a> .

**Details**

See the ‘[contourplot](#)’ function from the [lattice](#) package for plot customization options, and the functions [dssp](#) and [stride](#) for further details.

**Value**

Called for its effect.

**Note**

Be sure to check the correspondence of your ‘sse’ object with the ‘cij’ values being plotted as no internal checks are currently performed.

**Author(s)**

Barry Grant

## References

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

## See Also

[plot.bio3d](#), [plot.dmat](#), [filled.contour](#), [contour](#), [image](#) [plot.default](#), [dssp](#), [stride](#)

## Examples

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

## Read reference PDB and trim it to match the trajectory
pdb <- trim(read.pdb("1W5Y"), 'calpha')

## select residues 24 to 27 and 85 to 90 in both chains
inds <- atom.select(pdb, resno=c(24:27,85:90))

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

## Dynamic cross-correlations of atomic displacements
cij <- dccm(xyz)

## Default plot
plot.dccm(cij)

## Change the color scheme and the range of colored data levels
plot.dccm(cij, contour=FALSE, col.regions=bwr.colors(200), at=seq(-1,1,by=0.01) )

## Add secondary structure annotation to plot margins
plot.dccm(cij, sse=pdb)

## Add additional margin annotation for chains
## Also label x- and y-axis with PDB residue numbers
ch <- ifelse(pdb$atom$chain=="A", 1,2)
plot.dccm(cij, resno=pdb, sse=pdb, margin.segments=ch)

## Plot with cluster annotation from dynamic network analysis
#net <- cna(cij)
#plot.dccm(cij, margin.segments=net$raw.communities$membership)

## Focus on major communities (i.e. exclude those below a certain total length)
#plot.dccm(cij, margin.segments=net$raw.communities$membership, segment.min=25)

## End(Not run)
```

plot.dmat

*Plot Distance Matrix***Description**

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

**Usage**

```
## S3 method for class 'dmat'
plot(x, key = TRUE, resnum.1 = c(1:ncol(x)), resnum.2 = resnum.1,
      axis.tick.space = 20, zlim = range(x, finite = TRUE),
      nlevels = 20, levels = pretty(zlim, nlevels),
      color.palette = bwr.colors,
      col = color.palette(length(levels) - 1),
      axes = TRUE, key.axes, xaxs = "i", yaxs = "i", las = 1,
      grid = TRUE, grid.col = "yellow", grid.nx = floor(ncol(x)/30),
      grid.ny = grid.nx, center.zero = TRUE, flip=TRUE, ...)
```

**Arguments**

x	a numeric distance matrix generated by the function <a href="#">dm</a> .
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 <i>strictly</i> 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 flipped.
...	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/1hel.pdb", package="bio3d") )

# DM
d <- dm(pdb, "calpha")

# 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")

## Not run:
```

```

# Download and align two PDB files
pdbs <- pdbaln( get.pdb( c( "4q21", "521p"), path=tempdir(), overwrite=TRUE))

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

# Calculate DDM
c <- a - b

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

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

## End(Not run)

```

---

plot.enma

*Plot eNMA Results*


---

### Description

Produces a plot of atomic fluctuations of aligned normal modes.

### Usage

```

## S3 method for class 'enma'
plot(x,
      pdbs = NULL,
      xlab = NULL,
      ylab="Fluctuations", ...)

```

### Arguments

x	the results of ensemble NMA obtained with <code>nma.pdbs</code> . Alternatively, a matrix in the similar format as <code>enma\$fluctuations</code> can be provided.
pdbs	an object of class 'pdbs' in which the 'enma' object x was obtained from. If provided SSE data of the first structure of pdbs will drawn.
xlab	a label for the x axis.
ylab	labels for the y axes.
...	extra plotting arguments passed to <code>plot.fluct</code> that effect the atomic fluctuations plot only.

## Details

plot.enma produces a fluctuation plot of aligned nma objects. If corresponding pdbs object is provided the plot contains SSE annotation and appropriate residue index numbering.

## Value

Called for its effect.

## Author(s)

Lars Skjaerven, Barry Grant

## References

Skjaerven, L. et al. (2014) *BMC Bioinformatics* **15**, 399. Grant, B.J. et al. (2006) *Bioinformatics* **22**, 2695–2696.

## See Also

[nma](#), [plotb3](#), [plot.fluct](#)

## Examples

```
## Not run:
ids <- c("1a70_A", "1czp_A", "1frd_A", "1fxi_A", "1iue_A", "1pfd_A")
raw.files <- get.pdb(ids, path = "raw_pdb")
files <- pdbsplit(raw.files, ids, path = "raw_pdb/split_chain")

## Sequence/structure alignment
pdbs <- pdbaln(files)

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

## Plot fluctuations
plot(modes, pdbs=pdbs)

## Group and spread fluctuation profiles
hc <- hclust(as.dist(1-modes$rmsip))
col <- cutree(hc, k=2)
plot(modes, pdbs=pdbs, col=col, spread=TRUE)

## End(Not run)
```

---

`plot.fasta`*Plot a Multiple Sequence Alignment*

---

**Description**

Produces a schematic representation of a multiple sequence alignment.

**Usage**

```
## S3 method for class 'fasta'  
plot(x, hc = TRUE, labels = x$id, cex.lab = 0.7,  
      xlab = "Alignment index",  
      main = "Sequence Alignment Overview",  
      mar4 = 4, ...)
```

**Arguments**

<code>x</code>	multiple sequence alignment of class 'fasta' as obtained from <a href="#">seqaln</a> .
<code>hc</code>	logical, if TRUE plot a dendrogram on the left side. Alternatively, an object obtained from <a href="#">hclust</a> can be provided.
<code>labels</code>	labels corresponding to each row in the alignment.
<code>cex.lab</code>	scaling factor for the labels.
<code>xlab</code>	label for x-axis.
<code>main</code>	a main title for the plot.
<code>mar4</code>	margin size for the labels.
<code>...</code>	additional arguments passed to function <a href="#">hclust</a> .

**Details**

`plot.fasta` is a utility function for producing a schematic representation of a multiple sequence alignment.

**Value**

Called for its effect.

**Author(s)**

Lars Skjaerven

**References**

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

**See Also**

[seqaln](#), [read.fasta](#), [entropy](#), [aln2html](#).

**Examples**

```
# Read alignment
aln <- read.fasta(system.file("examples/kif1a.fa",package="bio3d"))

## alignment plot
plot(aln, labels=basename.pdb(aln$id))

## Works also for a 'pdbs' object
attach(transducin)
plot(pdbs)

detach(transducin)

## Not run:
infile <- "http://pfam.xfam.org/family/PF00071/alignment/seed/format?format=fasta"
aln <- read.fasta( infile )
plot(aln)

## End(Not run)
```

---

plot.fluct

*Plot Fluctuations*

---

**Description**

Produces a plot of atomic fluctuations obtained from ensemble normal mode analysis or molecular dynamics simulations.

**Usage**

```
## S3 method for class 'fluct'
plot(x,
      col = NULL, label = rownames(x), signif = FALSE,
      p.cutoff = 0.005, q.cutoff = 0.04,
      s.cutoff = 5, n.cutoff = 2, mean = FALSE, polygon = FALSE,
      spread = FALSE, offset = 1,
      ncore = NULL, ...)
```

**Arguments**

**x** a numeric vector or matrix containing atomic fluctuation data obtained from e.g. [nma.pdbs](#) or [rmsf](#).

**col** a character vector of plotting colors. Used also to group fluctuation profiles. NA values in col will omit the corresponding fluctuation profile in the plot.



label	a character vector of plotting labels with length matching nrow(x). If mean=TRUE, the length of label can be equal to the number of categories indicated by col.
signif	logical, if TRUE significance of fluctuation difference is calculated and annotated for each atomic position.
p.cutoff	Cutoff of p-value to define significance.
q.cutoff	Cutoff of the mean fluctuation difference to define significance.
s.cutoff	Cutoff of sample size in each group to calculate the significance.
n.cutoff	Cutoff of consecutive residue positions with significant fluctuation difference. If the actual number is less than the cutoff, corresponding positions will not be annotated.
mean	logical, if TRUE plot mean fluctuations of each group. Significance is still calculated with the original data.
polygon	logical, if TRUE a nicer plot with area under the line for the first row of x are filled with polygons.
ncore	number of CPU cores used to do the calculation. By default (ncore=NULL), use all available CPU cores. The argument is only used when signif=TRUE.
spread	logical, if TRUE the fluctuation profiles are spread - i.e. not on top of each other.
offset	numerical offset value in use when 'spread=TRUE'.
...	extra plotting arguments passed to plot.bio3d.

### Details

The significance calculation is performed when signif=TRUE and there are at least two groups with sample size larger than or equal to s.cutoff. A "two-sided" student's t-test is performed for each atomic position (each column of x). If x contains gaps, indicated by NAs, only non-gapped positions are considered. The position is considered significant if both p-value  $\leq$  p.cutoff and the mean value difference of the two groups, q, satisfies  $q \geq$  q.cutoff. If more than two groups are available, every pair of groups are subjected to the t-test calculation and the minimal p-value along with the q-value for the corresponding pair are used for the significance evaluation.

### Value

If significance is calculated, return a vector indicating significant positions.

### Author(s)

Xin-Qiu Yao, Lars Skjaerven, Barry Grant

### References

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

### See Also

[plot.bio3d](#), [rmsf](#), [nma.pdbs](#), [t.test](#), [polygon](#).

**Examples**

```

## Not run:
## load transducin example data
attach(transducin)

## subset of pdbs to analyze
inds = c(1:5, 16:20)
pdbs <- trim(pdb, row.inds=inds)
gaps.res = gap.inspect(pdb$ali)

## reference RESNO and SSE for axis annotations
resno <- pdbs$resno[1, gaps.res$f.inds]
sse <- pdbs$sse[1, gaps.res$f.inds]

## eNMA calculation and obtain modes of motion including atomic fluctuations
modes <- nma(pdb, ncore=NULL)
x = modes$fluctuation

## simple line plot with SSE annotation
plot.fluct(x, sse=sse, resno=resno)

## group data by specifying colors of each fluctuation line; same color indicates
## same group. Also do significance calculation and annotation
col = c(rep('red', 5), rep('blue', 5))
plot.fluct(x, col=col, signif=TRUE, sse=sse, resno=resno)

## spread lines
plot.fluct(x, col=col, signif=TRUE, sse=sse, resno=resno, typ='l', spread=TRUE)

## show only line of mean values for each group.
## Nicer plot with area shaded for the first group.
plot.fluct(x, col=col, signif=TRUE, sse=sse, resno=resno, mean=TRUE,
           polygon=TRUE, label=c('GTP', 'GDI'))

detach(transducin)

## End(Not run)

```

---

plot.geostas

*Plot Geostas Results*


---

**Description**

Plot an atomic movement similarity matrix with domain annotation

**Usage**

```

## S3 method for class 'geostas'
plot(x, at=seq(0, 1, 0.1), main="AMSM with Domain Assignment",
     col.regions=rev(heat.colors(200)),
     margin.segments=x$grps, ...)

```

**Arguments**

x	a object of type geostas as obtained by the 'geostas' function.
at	numeric vector specifying the levels to be colored.
main	a main title for the plot.
col.regions	color vector. See contourplot for more information.
margin.segments	a numeric vector of cluster membership as obtained from cutree() or other community detection method. This will be used for bottom and left margin annotation.
...	additional graphical parameters for <a href="#">plot.dccm</a> and contourplot.

**Details**

This is a wrapper function for [plot.dccm](#) with appropriate adjustments for plotting atomic movement similarity matrix obtained from function [geostas](#).

See the [plot.dccm](#) for more details.

**Value**

Called for its effect.

**Author(s)**

Barry Grant, Lars Skjaerven

**References**

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

**See Also**

[plot.dccm](#), [geostas](#)

---

plot.hmmmer

*Plot a Summary of HMMER Hit Statistics.*

---

**Description**

Produces a number of basic plots that should facilitate hit selection from the match statistics of a HMMER result.

**Usage**

```
## S3 method for class 'hmmmer'  
plot(x, ...)
```

**Arguments**

x                    HMMER results as obtained from the function `hmmer`.  
...                  arguments passed to `plot.blast`.

**Details**

See `plot.blast` for details.

**Value**

Produces a plot on the active graphics device and returns a three component list object:

hits	an ordered matrix detailing the subset of hits with a normalized score above the chosen cutoff. Database identifiers are listed along with their cluster group number.
acc	a character vector containing the database accession identifier of each hit above the chosen threshold.
pdb.id	a character vector containing the database accession identifier of each hit above the chosen threshold.
inds	a numeric vector containing the indices of the hits relative to the input hmmer object.

**Author(s)**

Barry Grant, Lars Skjaerven

**References**

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

**See Also**

`hmmer`, `blast.pdb`

**Examples**

```
## Not run:  
# HMMER server connection required - testing excluded  
  
##- PHMMER  
seq <- get.seq("2abl_A", outfile = tempfile())  
res <- hmmer(seq, db="pdb")  
plot.hmmer(res)  
  
## End(Not run)
```

---

plot.matrix.loadings *Plot Residue-Residue Matrix Loadings*

---

### Description

Plot residue-residue matrix loadings of a particular PC that is obtained from a principal component analysis (PCA) of cross-correlation or distance matrices.

### Usage

```
## S3 method for class 'matrix.loadings'  
plot(x, pc = 1, resno = NULL, sse = NULL,  
     mask.n = 0, plot = TRUE, ...)
```

### Arguments

x	the results of PCA as obtained from <a href="#">pca.array</a> .
pc	the principal component along which the loadings will be shown.
resno	numerical vector or 'pdb' object as obtained from <a href="#">read.pdb</a> to show residue number on the x- and y-axis.
sse	a 'sse' object as obtained from <a href="#">dssp</a> or <a href="#">stride</a> , or a 'pdb' object as obtained from <a href="#">read.pdb</a> to show secondary structural elements along x- and y-axis.
mask.n	the number of elements from the diagonal to be masked from output.
plot	logical, if FALSE no plot will be shown.
...	additional arguments passed to <a href="#">plot.dccm</a> .

### Details

The function plots loadings (the eigenvectors) of PCA performed on a set of matrices such as distance matrices from an ensemble of crystallographic structures and residue-residue cross-correlations or covariance matrices derived from ensemble NMA or MD simulation replicates (See [pca.array](#) for detail). Loadings are displayed as a matrix with dimension the same as the input matrices of the PCA. Each element of loadings represents the proportion that the corresponding residue pair contributes to the variance in a particular PC. The plot can be used to identify key regions that best explain the variance of underlying matrices.

### Value

Plot and also returns a numeric matrix containing the loadings.

### Author(s)

Xin-Qiu Yao

## References

Skjaerven, L. et al. (2014) *BMC Bioinformatics* **15**, 399. Grant, B.J. et al. (2006) *Bioinformatics* **22**, 2695–2696.

## See Also

[plot.dccm](#), [pca.array](#)

## Examples

```
## Not run:
attach(transducin)
gaps.res <- gap.inspect(pdb$ali)
sse <- pdb$sse[1, gaps.res$f.inds]

# calculate modes
modes <- nma(pdb, ncore=NULL)

# calculate cross-correlation matrices from the modes
cijs <- dcm(modes, ncore=NULL)$all.dcm

# do PCA on cross-correlation matrices
pc <- pca.array(cijs)

# plot loadings
l <- plot.matrix.loadings(pc, sse=sse)
l[1:10, 1:10]

# plot loadings with elements 10-residue separated from diagonal masked
plot.matrix.loadings(pc, sse=sse, mask.n=10)

## End(Not run)
```

---

plot.nma

*Plot NMA Results*

---

## Description

Produces eigenvalue/frequency spectrum plots and an atomic fluctuations plot.

## Usage

```
## S3 method for class 'nma'
plot(x, pch = 16, col = par("col"), cex=0.8, mar=c(6, 4, 2, 2),...)
```

**Arguments**

x	the results of normal modes analysis obtained with <a href="#">nma</a> .
pch	a vector of plotting characters or symbols: see <a href="#">points</a> .
col	a character vector of plotting colors.
cex	a numerical single element vector giving the amount by which plotting text and symbols should be magnified relative to the default.
mar	A numerical vector of the form c(bottom, left, top, right) which gives the number of lines of margin to be specified on the four sides of the plot.
...	extra plotting arguments passed to <a href="#">plot.bio3d</a> that effect the atomic fluctuations plot only.

**Details**

`plot.nma` produces an eigenvalue (or frequency) spectrum plot together with a plot of the atomic fluctuations.

**Value**

Called for its effect.

**Author(s)**

Lars Skjaerven

**References**

Skjaerven, L. et al. (2014) *BMC Bioinformatics* **15**, 399. Grant, B.J. et al. (2006) *Bioinformatics* **22**, 2695–2696.

**See Also**

[nma](#), [plot.bio3d](#)

**Examples**

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

## Calculate modes
modes <- nma(pdb)

plot(modes, sse=pdb)
```

plot.pca

*Plot PCA Results***Description**

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

**Usage**

```
## S3 method for class 'pca'
plot(x, pc.axes=NULL, pch=16, col=par("col"), cex=0.8, mar=c(4, 4, 1, 1),...)

## S3 method for class 'pca.scree'
plot(x, y = NULL, type = "o", pch = 18,
      main = "", sub = "", xlim = c(0, 20), ylim = NULL,
      ylab = "Proportion of Variance (%)",
      xlab = "Eigenvalue Rank", axes = TRUE, ann = par("ann"),
      col = par("col"), lab = TRUE, ...)

## S3 method for class 'pca.score'
plot(x, inds=NULL, col=rainbow(nrow(x)), lab = "", ...)
```

**Arguments**

x	the results of principal component analysis obtained with <a href="#">pca.xyz</a> .
pc.axes	an optional numeric vector of length two specifying the principal components to be plotted. A NULL value will result in an overview plot of the first three PCs and a scree plot. See examples.
pch	a vector of plotting characters or symbols: see 'points'.
col	a character vector of plotting colors.
cex	a numerical single element vector giving the amount by which plotting text and symbols should be magnified relative to the default.
mar	A numerical vector of the form c(bottom, left, top, right) which gives the number of lines of margin to be specified on the four sides of the plot.
inds	row indices of the conformers to label.
lab	a character vector of plot labels.
y	the y coordinates for the scree plot.
type	one-character string giving the type of plot desired.
main	a main title for the plot, see also 'title'.
sub	a sub-title for the plot.
xlim	the x limits of the plot.
ylim	the y limits of the plot.
ylab	a label for the y axis.



xlab	a label for the x axis.
axes	a logical value indicating whether both axes should be drawn.
ann	a logical value indicating whether the default annotation (title and x and y axis labels) should appear on the plot.
...	extra plotting arguments.

### Details

plot.pca is a wrapper calling both plot.pca.score and plot.pca.scrie resulting in a 2x2 plot with three score plots and one scree plot.

### Value

Produces a plot of PCA results in the active graphics device and invisibly returns the plotted 'z' coordinates along the requested 'pc.axes'. See examples section where these coordinates are used to identify plotted points.

### Author(s)

Barry Grant

### References

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

### See Also

[pca.xyz](#), [plot.bio3d](#)

### Examples

```
attach(transducin)

pc.xray <- pca(pdb$xyz, rm.gaps=TRUE)
plot(pc.xray)

## Color plot by nucleotide state
vcolors <- annotation[, "color"]
plot(pc.xray, col=vcolors)

## Focus on a single plot of PC1 vs PC2
x <- plot(pc.xray, pc.axes=1:2, col=vcolors)

## Identify points interactively with mouse clicks
#identify(x, labels=basename.pdb(pdb$id))

## Add labels to select points
inds <- c(1,10,37)
text(x[inds,], labels=basename.pdb(pdb$id[inds]), col="blue")

## Alternative labeling method
```

```
#labs <- rownames(annotation)
#inds <- c(2,7)
#plot.pca.score(pc.xray, inds=inds, col=vcolors, lab=labs)

## color by seq identity groupings
#ide <- seqidentity(pdb$ali)
#hc <- hclust(as.dist(1-ide))
#grps <- cutree(hc, h=0.2)
#vcolors <- rainbow(max(grps))[grps]
#plot.pca.score(pc.xray, inds=inds, col=vcolors, lab=labs)

detach(transducin)
```

---

plot.pca.loadings      *Plot Residue Loadings along PC1 to PC3*

---

### Description

Plot residue loadings along PC1 to PC3 from a given xyz C-alpha matrix of loadings.

### Usage

```
## S3 method for class 'pca.loadings'
plot(x, resnums = seq(1, (length(x[, 1])/3), 25), ...)
```

### Arguments

x	the results of principal component analysis obtained from <a href="#">pca.xyz</a> , or just the loadings returned from <a href="#">pca.xyz</a> .
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**

```
attach(transducin)

pc.xray <- pca.xyz(pdb$xyz[, gap.inspect(pdb$xyz)$f.inds])
plot.pca.loadings(pc.xray$U)

detach(transducin)
```

---

plot.rmsip

*Plot RMSIP Results*

---

**Description**

Produces a heat plot of RMSIP (Root mean square inner product) for the visualization of modes similarity.

**Usage**

```
## S3 method for class 'rmsip'
plot(x, xlab = NULL, ylab = NULL, col = gray(50:0/50),
     zlim=c(0,1), ...)
```

**Arguments**

x	an object of class rmsip.
xlab	a label for the x axis, defaults to 'a'.
ylab	a label for the y axis, defaults to 'b'.
col	a vector of colors for the RMSIP map (or overlap values).
zlim	the minimum and maximum 'z' values for which colors should be plotted.
...	additional arguments to function image.

**Details**

plot.rmsip produces a color image with the function image.

**Value**

Called for its effect.

**Author(s)**

Lars Skjaerven

**References**

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

**See Also**

[rmsip](#), [overlap](#), [nma](#), [image](#).

**Examples**

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

## Perform NMA
modes.a <- nma(pdb, ff="calpha")
modes.b <- nma(pdb, ff="anm")

## Calculate and plot RMSIP
r <- rmsip(modes.a, modes.b)
plot(r)
```

---

print.cna

*Summarize and Print Features of a cna Network Graph*


---

**Description**

These functions attempt to summarize and print a cna network graph to the terminal in a human readable form.

**Usage**

```
## S3 method for class 'cna'
print(x, ...)
## S3 method for class 'cna'
summary(object, verbose=TRUE, ...)
```

**Arguments**

x	A cna network and community object as obtained from the function 'cna'.
object	A cna network and community object as obtained from the function 'cna'.
verbose	Logical, if TRUE a community summary table is printed to screen.
...	Extra arguments passed to the 'write.table' function.

**Details**

Simple summary and print methods for protein dynamic networks.

**Value**

The function `summary.cna` returns a list with the following components:

<code>id</code>	A community number/identifier vector with an element for each community.
<code>size</code>	A numeric community size vector, with elements giving the number of nodes within each community.
<code>members</code>	A list detailing the nodes within each community.

**Author(s)**

Guido Scarabelli and Barry Grant

**References**

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

**See Also**

[cna](#), [print.igraph](#), [str.igraph](#), [igraph.plotting](#)

**Examples**

```
if (!requireNamespace("igraph", quietly = TRUE)) {
  message('Need igraph installed to run this example')
} else {

  ## Load the correlation network
  attach(hivp)

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

  ## Examine network composition
  print(net)
  x<- summary(net)
  x$members[[2]]

  detach(hivp)
}
```

---

`print.core`*Printing Core Positions and Returning Indices*

---

**Description**

Print method for `core.find` objects.

**Usage**

```
## S3 method for class 'core'  
print(x, vol = NULL, ...)
```

**Arguments**

<code>x</code>	a list object obtained with the function <code>core.find</code> .
<code>vol</code>	the maximal cumulative volume value at which core positions are detailed.
<code>...</code>	additional arguments to ‘print’.

**Value**

Returns a three component list of indices:

<code>atom</code>	atom indices of core positions
<code>xyz</code>	xyz indices of core positions
<code>resno</code>	residue numbers of core positions

**Note**

The produced `plot.core` function can be useful for deciding on the core/non-core boundary.

**Author(s)**

Barry Grant

**References**

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

**See Also**

[core.find](#), [plot.core](#)

**Examples**

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

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

##-- Fit on these relatively invariant subset of positions
core.inds <- print(core, vol=0.5)

print(core, vol=0.7)
print(core, vol=1.0)

## End(Not run)
```

---

print.fasta

*Printing Sequence Alignments*


---

**Description**

Print method for fasta and pdbs sequence alignment objects.

**Usage**

```
## S3 method for class 'fasta'
print(x, alignment=TRUE, ...)
.print.fasta.ali(x, width = NULL, col.inds = NULL, numbers = TRUE,
                 conservation=TRUE, ...)
```

**Arguments**

x	a sequence alignment object as obtained from the functions <a href="#">read.fasta</a> , <a href="#">read.fasta.pdb</a> , <a href="#">pdbaln</a> , <a href="#">seqaln</a> , etc.
alignment	logical, if TRUE the sequence alignment will be printed to screen.
width	a single numeric value giving the number of residues per printed sequence block. By default this is determined from considering alignment identifier widths given a standard 85 column terminal window.
col.inds	an optional numeric vector that can be used to select subsets of alignment positions/columns for printing.
numbers	logical, if TRUE position numbers and a tick-mark every 10 positions are printed above and below sequence blocks.

conservation logical, if TRUE conserved and semi-conserved columns in the alignment are marked with an '\*' and '^', respectively.  
... additional arguments to '.print.fasta.ali'.

**Value**

Called mostly for its effect but also silently returns block divided concatenated sequence strings as a matrix.

**Author(s)**

Barry Grant

**References**

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

**See Also**

[read.fasta](#), [read.fasta.pdb](#), [pdbaln](#), [seqaln](#)

**Examples**

```
file <- system.file("examples/kif1a.fa", package="bio3d")
aln <- read.fasta(file)
print(aln)

# print(aln, col.inds=30:100, numbers=FALSE)
```

---

print.xyz

*Printing XYZ coordinates*

---

**Description**

Print method for objects of class 'xyz'.

**Usage**

```
## S3 method for class 'xyz'
print(x, ...)
```

**Arguments**

x a 'xyz' object indicating 3-D coordinates of biological molecules.  
... additional arguments passed to 'print'.



**Value**

Called for its effect.

**Author(s)**

Barry Grant

**References**

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

**See Also**

[is.xyz](#), [read.ncdf](#), [read.pdb](#), [read.dcd](#), [fit.xyz](#)

**Examples**

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

---

project.pca

*Project Data onto Principal Components*

---

**Description**

Projects data onto principal components.

**Usage**

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

**Arguments**

data	a numeric vector or row-wise matrix of data to be projected.
pca	an object of class "pca" as obtained from functions <code>pca.xyz</code> or <code>pca.tor</code> .
angular	logical, if TRUE the data to be projected is treated as torsion angle data.
fit	logical, if TRUE the data is first fitted to <code>pca\$mean</code> .
...	other parameters for <a href="#">fit.xyz</a> .
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 original data projected onto the PCs) for conversion to xyz coordinates.

**Value**

A numeric vector or matrix of projected PC scores.

**Author(s)**

Karim ElSawy and Barry Grant

**References**

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

**See Also**

[pca.xyz](#), [pca.tor](#), [fit.xyz](#)

**Examples**

```
## Not run:
attach(transducin)

gaps.pos <- gap.inspect(pdb$xyz)

#-- Do PCA without structures 2 and 7
pc.xray <- pca.xyz(pdb$xyz[-c(2,7), gaps.pos$f.inds])

#-- Project structures 2 and 7 onto the PC space
d <- project.pca(pdb$xyz[c(2,7), gaps.pos$f.inds], pc.xray)

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

detach(transducin)

## End(Not run)
```

---

prune.cna

*Prune A cna Network Object*

---

**Description**

Remove nodes and their associated edges from a cna network graph.

**Usage**

```
prune.cna(x, edges.min = 1, size.min = 1)
```

**Arguments**

x	A protein network graph object as obtained from the 'cna' function.
edges.min	A single element numeric vector specifying the minimum number of edges that retained nodes should have. Nodes with less than 'edges.min' will be pruned.
size.min	A single element numeric vector specifying the minimum node size that retained nodes should have. Nodes with less composite residues than 'size.min' will be pruned.

**Details**

This function is useful for cleaning up cna network plots by removing, for example, small isolated nodes. The output is a new cna object minus the pruned nodes and their associated edges. Node naming is preserved.

**Value**

A cna class object, see function [cna](#) for details.

**Note**

Some improvements to this function are required, including a better effort to preserve the original community structure rather than calculating a new one. Also may consider removing nodes from the raw.network object that is returned also.

**Author(s)**

Barry Grant

**References**

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

**See Also**

[cna](#), [summary.cna](#), [vmd.cna](#), [plot.cna](#)

**Examples**

```
if (!requireNamespace("igraph", quietly = TRUE)) {
  message('Need igraph installed to run this example')
} else {

# Load the correlation network
attach(hivp)

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

```
# Plot coarse grain network based on dynamically coupled communities
par(mfcol=c(1,2), mar=c(0,0,0,0))
plot.cna(net)

# Prune network
dnet <- prune.cna(net, edges.min = 1)
plot(dnet)

detach(hivp)

}
```

---

pymol

*Biomolecular Visualization with PyMOL*

---

## Description

Visualize Bio3D structure objects in PyMOL

## Usage

```
pymol(...)

## S3 method for class 'pdbs'
pymol(pdbs, col=NULL, as="ribbon", file=NULL, type="script",
      exefile="pymol", user.vec=NULL, ...)

## S3 method for class 'nma'
pymol(...)

## S3 method for class 'pca'
pymol(...)

## S3 method for class 'modes'
pymol(modes, mode=NULL, file=NULL, scale=5, dual=FALSE,
      type="script", exefile="pymol", ...)

## S3 method for class 'dccm'
pymol(dccm, pdb, file=NULL,
      step=0.2, omit=0.2, radius = 0.15,
      type="script", exefile="pymol", ...)
```

## Arguments

pdbs                   aligned C-alpha Cartesian coordinates as obtained with [read.fasta.pdb](#) or [pdbaln](#).

<code>col</code>	a single element character vector specifying the coloring of the structures. Options are: 'index', 'index2', 'gaps', 'rmsf', 'user'. Special cases: Provide a 'core' object as obtained by <code>core.find</code> to color on the invariant core. Alternatively, provide a vector containing the color code for each structure in the 'pdb' object.
<code>user.vec</code>	User defined vector for coloring. Only used if <code>col="user"</code> .
<code>as</code>	show as 'ribbon', 'cartoon', 'lines', 'putty'.
<code>file</code>	a single element character vector specifying the file name of the PyMOL session/script file.
<code>type</code>	a single element character vector specifying the output type: 'script' generates a .pml script; 'session' generates a .pse session file; 'launch' launches pymol.
<code>exefile</code>	file path to the 'PYMOL' program on your system (i.e. how is 'PYMOL' invoked). If NULL, use OS-dependent default path to the program.
<code>modes</code>	an object of class <code>nma</code> or <code>pca</code> as obtained from functions <code>nma</code> or <code>pca.xyz</code> .
<code>mode</code>	the mode number for which the vector field should be made.
<code>scale</code>	global scaling factor.
<code>dual</code>	logical, if TRUE mode vectors are also drawn in both direction.
<code>dccm</code>	an object of class <code>dccm</code> as obtained from function <code>dccm</code> .
<code>pdb</code>	an object of class <code>pdb</code> as obtained from function <code>read.pdb</code> or a numerical vector of Cartesian coordinates.
<code>step</code>	binning interval of cross-correlation coefficients.
<code>omit</code>	correlation coefficients with values (0-omit, 0+omit) will be omitted from visualization.
<code>radius</code>	numeric, radius of visualized correlation cylinders in PyMol. Alternatively, a matrix with the same dimensions as <code>dccm</code> can be provided, e.g. to draw cylinders with radii associated to the pairwise correlation value.
<code>...</code>	arguments passed to function <code>pymol.modes</code> for 'nma' and 'pca' objects.

## Details

These functions provides a convenient approach for the visualization of Bio3D objects in PyMOL. See examples for more details.

**DCCM PyMOL visualization:** This function generates a PyMOL (python) script that will draw colored lines between (anti)correlated residues. The PyMOL script file is stored in the working directory with filename "R.py". PyMOL will only be launched (and opened) when using argument 'type='launch''. Alternatively a PDB file with CONECT records will be generated (when argument `type='pdb'`).

For the PyMOL version, PyMOL CGO objects are generated - each object representing a range of correlation values (corresponding to the actual correlation values as found in the correlation matrix). E.g. the PyMOL object with name "cor\_-1\_-08" would display all pairs of correlations with values between -1 and -0.8.

**NMA / PCA PyMOL vector field visualization:** This function generates a PyMOL (python) script for drawing mode vectors on a PDB structure. The PyMOL script file is stored in the working directory with filename "R.py".

**Value**

Called for its action

**Author(s)**

Lars Skjaerven

**References**

Skjaerven, L. et al. (2014) *BMC Bioinformatics* **15**, 399. Grant, B.J. et al. (2006) *Bioinformatics* **22**, 2695–2696.

**See Also**

view

**Examples**

```
## Not run:

##- pymol with a 'pdb' object
attach(transducin)

# build a pymol session containing all structures in the PDBs object
pymol(pdb)

# color by invariant core (
# core <- core.find(pdb)
pymol(pdb, col=core)

# color by RMSF
pymol(pdb, col="rmsf")

# color by a user defined vector
# For example, colored by averaged contact density around each residue
cm <- cmap(pdb, binary=FALSE)
vec <- rowSums(cm, na.rm=TRUE)
pymol(pdb, col="user", user.vec=vec)

# color by clustering
rd <- rmsd(pdb$xyz)
hc <- hclust(as.dist(rd))
grps <- cutree(hc, k=3)
pymol(pdb, col=grps)

##- pymol with a 'dcm' object
## Fetch stucture
pdb <- read.pdb( system.file("examples/1hel.pdb", package="bio3d") )

## Calculate normal modes
```

```
modes <- nma(pdb)

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

pymol(cm, modes$xyz)

##- pymol with a 'nma' or 'pca' object
pymol(modes, mode=7)

detach(transducin)

## End(Not run)
```

---

read.all

*Read Aligned Structure Data*

---

### Description

Read aligned PDB structures and store their equalvalent atom data, including xyz coordinates, residue numbers, residue type and B-factors.

### Usage

```
read.all(aln, prefix = "", pdbext = "", sel = NULL, rm.wat=TRUE, rm.ligand=FALSE,
         compact = TRUE, ncore = NULL, ...)
```

### Arguments

aln	an alignment data structure obtained with <a href="#">read.fasta</a> .
prefix	prefix to aln\$id to locate PDB files.
pdbext	the file name extension of the PDB files.
sel	a selection string detailing the atom type data to store (see function store.atom)
rm.wat	logical, if TRUE water atoms are removed.
rm.ligand	logical, if TRUE ligand atoms are removed.
compact	logical, if TRUE the number of atoms stored for each aligned residue varies according to the amino acid type. If FALSE, the constant maximum possible number of atoms are stored for all aligned residues.
ncore	number of CPU cores used to do the calculation. By default (ncore=NULL) use all detected CPU cores.
...	other parameters for <a href="#">read.pdb</a> .

### Details

The input `aln`, produced with [read.fasta](#), must have identifiers (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 identifier ‘`mypdbdir/1bg2.pdb`’ or ‘`1bg2`’ if input ‘`prefix`’ and ‘`pdtext`’ equal ‘`mypdb-dir/`’ and ‘`pdb`’. See the examples below.

Sequence miss-matches will generate errors. Thus, care should be taken to ensure that the sequences in the alignment match the sequences in their associated PDB files.

### Value

Returns a list of class “`pdbs`” with the following five components:

<code>xyz</code>	numeric matrix of aligned C-alpha coordinates.
<code>resno</code>	character matrix of aligned residue numbers.
<code>b</code>	numeric matrix of aligned B-factor values.
<code>chain</code>	character matrix of aligned chain identifiers.
<code>id</code>	character vector of PDB sequence/structure names.
<code>ali</code>	character matrix of aligned sequences.
<code>resid</code>	character matrix of aligned 3-letter residue names.
<code>all</code>	numeric matrix of aligned equalvalent atom coordinates.
<code>all.elety</code>	numeric matrix of aligned atom element types.
<code>all.resid</code>	numeric matrix of aligned three-letter residue codes.
<code>all.resno</code>	numeric matrix of aligned residue numbers.
<code>all.grpby</code>	numeric vector indicating the group of atoms belonging to the same aligned residue.
<code>all.hetatm</code>	a list of ‘ <code>pdb</code> ’ objects for non-protein atoms.

### Note

This function is still in development and is NOT part of the official `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 storing all data for 'sel'
sel <- c("N", "CA", "C", "O", "CB", "*G", "*D", "*E", "*Z")
pdbs <- read.all(aln, sel=sel)

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

## Fit structures
nxyz <- fit.xyz(pdbs$all[1,], pdbs$all,
               fixed.inds = core.ind,
               mobile.inds = core.ind)

ngap.col <- gap.inspect(nxyz)

#npc.xray <- pca.xyz(nxyz[,ngap.col$f.inds])

#a <- mktrj.pca(npc.xray, pc=1, file="pc1-all.pdb",
#             eley=pdb$all.eley[1,unique( ceiling(ngap.col$f.inds/3) )],
#             resid=pdb$all.resid[1,unique( ceiling(ngap.col$f.inds/3) )],
#             resno=pdb$all.resno[1,unique( ceiling(ngap.col$f.inds/3) )] )

## End(Not run)

```

---

read.cif

*Read mmCIF File*


---

**Description**

Read a Protein Data Bank (mmCIF) coordinate file.

**Usage**

```

read.cif(file, maxlines = -1, multi = FALSE,
         rm.insert = FALSE, rm.alt = TRUE, verbose = TRUE)

```

**Arguments**

**file** a single element character vector containing the name of the mmCIF file to be read, or the four letter PDB identifier for online file access.

maxlines	the maximum number of lines to read before giving up with large files. By default it will read up to the end of input on the connection.
multi	logical, if TRUE multiple ATOM records are read for all models in multi-model files and their coordinates returned.
rm.insert	logical, if TRUE PDB insert records are ignored.
rm.alt	logical, if TRUE PDB alternate records are ignored.
verbose	print details of the reading process.

### Details

The current version of `read.cif` reads only ATOM/HETATM records and creates a `pdb` object of the data.

See `read.pdb` for more info.

### Value

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

atom	a data.frame containing all atomic coordinate ATOM and HETATM data, with a row per ATOM/HETATM and a column per record type. See below for details of the record type naming convention (useful for accessing columns).
xyz	a numeric matrix of class "xyz" containing the ATOM and HETATM coordinate data.
calpha	logical vector with length equal to <code>nrow(atom)</code> with TRUE values indicating a C-alpha "elety".
call	the matched call.

### Author(s)

Lars Skjaerven

### References

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

### See Also

[read.pdb](#), [atom.select](#), [write.pdb](#), [trim.pdb](#), [cat.pdb](#), [read.prmtop](#), [as.pdb](#), [read.dcd](#), [read.ncdf](#),

### Examples

```
## Read a mmCIF file from the RCSB online database
# cif <- read.cif("1hel")
```

---

`read.crd`*Read Coordinate Data from Amber or Charmm*

---

### Description

Read a CHARMM CARD (CRD) or AMBER coordinate file.

### Usage

```
read.crd(file, ...)
```

### Arguments

<code>file</code>	the name of the coordinate file to be read.
<code>...</code>	additional arguments passed to the methods <code>read.crd.charmm</code> or <code>read.crd.amber</code> .

### Details

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

[read.crd.charmm](#) will be used for file extension `'crd'`.

[read.crd.amber](#) will be used for file extension `'rst'` or `'inpcrd'`.

See examples for each corresponding function for more details.

### Value

See the `'value'` section for the corresponding functions for more details.

### Author(s)

Barry Grant and Lars Skjaerven

### References

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

### See Also

[read.crd.amber](#), [read.crd.charmm](#), [write.crd](#), [read.prmtop](#), [read.pdb](#), [write.pdb](#), [atom.select](#), [read.dcd](#), [read.ncdf](#)

## Examples

```
## Not run:
## Read a PRMTOP file
prmtop <- read.prmtop(system.file("examples/crambin.prmtop", package="bio3d"))
print(prmtop)

## Read a Amber CRD file
crds <- read.crd(system.file("examples/crambin.inpcrd", package="bio3d"))

## Atom selection
ca.indes <- atom.select(prmtop, "calpha")

## Convert to PDB format
pdb <- as.pdb(prmtop, crds, indes=ca.indes)

## End(Not run)
```

---

read.crd.amber	<i>Read AMBER Coordinate files</i>
----------------	------------------------------------

---

## Description

Read coordinate data from an AMBER coordinate / restart file.

## Usage

```
## S3 method for class 'amber'
read.crd(file, ...)
```

## Arguments

file	name of crd file to read.
...	arguments passed to and from functions.

## Details

Read a AMBER Coordinate format file.

## Value

A list object of type 'amber' and 'crd' with the following components:

xyz	a numeric matrix of class 'xyz' containing the Cartesian coordinates.
velocities	a numeric vector containing the atom velocities.
time	numeric, length of the simulation (applies to Amber restart coordinate files).
natoms	total number of atoms in the coordinate file.
box	dimensions of the box.

**Note**

See AMBER documentation for Coordinate format description.

**Author(s)**

Lars Skjaerven

**References**

Grant, B.J. et al. (2006) *Bioinformatics* **22**, 2695–2696. <http://ambermd.org/FileFormats.php>

**See Also**

[read.prmtop](#), [read.ncdf](#), [as.pdb](#), [atom.select](#), [read.pdb](#), [read.crd.charmm](#)

**Examples**

```
## Not run:
## Read Amber PRMTOP and CRD files
prm <- read.prmtop(system.file("examples/crambin.prmtop", package="bio3d"))
crd <- read.crd(system.file("examples/crambin.inpcrd", package="bio3d"))

## Convert to PDB format
pdb <- as.pdb(prm, crd)

## Atom selection
ca.inds <- atom.select(prm, "calpha")

## End(Not run)
```

---

read.crd.charmm	<i>Read CRD File</i>
-----------------	----------------------

---

**Description**

Read a CHARMM CARD (CRD) coordinate file.

**Usage**

```
## S3 method for class 'charmm'
read.crd(file, ext = TRUE, verbose = TRUE, ...)
```

**Arguments**

file	the name of the CRD file to be read.
ext	logical, if TRUE assume expanded CRD format.
verbose	print details of the reading process.
...	arguments going nowhere.

## 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 <code>nrow(atom)</code> with TRUE values indicating a C-alpha "eley".

## 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 "eley", 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:

[https://www.charmm.org/wiki//index.php/CHARMM:The\\_Basics](https://www.charmm.org/wiki//index.php/CHARMM:The_Basics).

## See Also

[write.crd](#), [read.pdb](#), [atom.select](#), [write.pdb](#), [read.dcd](#), [read.fasta.pdb](#), [read.fasta](#)

## Examples

```
## Not run:
pdb <- read.pdb("1bg2")
crdfile <- paste(tempfile(), '.crd', sep='')
write.crd(pdb, file=crdfile)
crd <- read.crd(crdfile, ext=FALSE)
ca.inds <- which(crd$calpha)
crd$atom[ca.inds[1:20],c("x","y","z")]
# write.pdb(crd, file=tempfile())

## End(Not run)
```

---

`read.dcd`*Read CHARMM/X-PLOR/NAMD Binary DCD files*

---

**Description**

Read coordinate data from a binary DCD trajectory file.

**Usage**

```
read.dcd(trjfile, big=FALSE, verbose = TRUE, cell = FALSE)
```

**Arguments**

<code>trjfile</code>	name of trajectory file to read. A vector if treat a batch of files
<code>big</code>	logical, if TRUE attempt to read large files into a <code>big.matrix</code> object
<code>verbose</code>	logical, if TRUE print details of the reading process.
<code>cell</code>	logical, if TRUE return cell information only. Otherwise, return coordinates.

**Details**

Reads a CHARMM or X-PLOR/NAMD binary trajectory file with either big- or little-endian storage formats.

Reading is accomplished with two different sub-functions: `dcd.header`, which reads header info, and `dcd.frame`, which takes header information and reads atoms frame by frame producing an `nframes/natom*3` matrix of cartesian coordinates or an `nframes/6` matrix of cell parameters.

**Value**

A numeric matrix of xyz coordinates with a frame/structure per row and a Cartesian coordinate per column or a numeric matrix of cell information with a frame/structure per row and lengths and angles per column.

**Note**

See CHARMM documentation for DCD format description.

If you experience problems reading your trajectory file with `read.dcd()` consider first reading your file into VMD and from there exporting a new DCD trajectory file with the 'save coordinates' option. This new file should be easily read with `read.dcd()`.

Error messages beginning 'cannot allocate vector of size' indicate a failure to obtain memory, either because the size exceeded the address-space limit for a process or, more likely, because the system was unable to provide the memory. Note that on a 32-bit OS there may well be enough free memory available, but not a large enough contiguous block of address space into which to map it. In such cases try setting the input option 'big' to TRUE. This is an experimental option that results in a 'big.matrix' object.

**Author(s)**

Barry Grant

**References**Grant, B.J. et al. (2006) *Bioinformatics* **22**, 2695–2696.**See Also**[read.pdb](#), [write.pdb](#), [atom.select](#)**Examples**

```
# Redundant testing excluded

##-- Read cell parameters from example trajectory file
trtfile <- system.file("examples/hivp.dcd", package="bio3d")
trj <- read.dcd(trtfile, cell = TRUE)
##-- Read coordinates from example trajectory file
trj <- read.dcd(trtfile)

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

## select residues 24 to 27 and 85 to 90 in both chains
inds <- atom.select(pdb, resno=c(24:27,85:90), elety='CA')

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

##-- RMSD of trj frames from PDB
r1 <- rmsd(a=pdb, b=xyz)

## Not run:
# Pairwise RMSD of trj frames for positions 47 to 54
flap.inds <- atom.select(pdb, resno=c(47:54), elety='CA')
p <- rmsd(xyz[,flap.inds$xyz])
# plot highlighting flap opening?
plot.dmat(p, color.palette = mono.colors)

## End(Not run)
```



---

read.fasta	<i>Read FASTA formatted Sequences</i>
------------	---------------------------------------

---

**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 identifiers.
call	the matched call.

**Note**

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

**Author(s)**

Barry Grant

**References**

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

**See Also**

[read.fasta.pdb](#)

**Examples**

```

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

# Print alignment overview
aln

# Sequence names/ids
head( aln$id )

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

# Sequence d2a4f_b
aa123( aln$ali["d2a4f_b",] )

# Write out positions 33 to 45 only
#aln$ali=aln$ali[,30:45]
#write.fasta(aln, file="eg2.fa")

```

---

read.fasta.pdb

*Read Aligned Structure Data*


---

**Description**

Read aligned PDB structures and store their C-alpha atom data, including xyz coordinates, residue numbers, residue type and B-factors.

**Usage**

```

read.fasta.pdb(aln, prefix = "", pdbext = "", fix.ali = FALSE,
               pdblast=NULL, ncore = 1, nseg.scale = 1, progress = NULL, ...)

```

**Arguments**

aln	an alignment data structure obtained with <a href="#">read.fasta</a> .
prefix	prefix to aln\$id to locate PDB files.
pdbext	the file name extention of the PDB files.
fix.ali	logical, if TRUE check consistence between \$ali and \$resno, and correct \$ali if they don't match.
pdblist	an optional list of pdb objects with sequence corresponding to the alignments in aln. Primarily used through function pdbaln when the PDB objects already exists (avoids reading PDBs from file).
ncore	number of CPU cores used to do the calculation. ncore>1 requires package 'parallel' installed.

nseg.scale	split input data into specified number of segments prior to running multiple core calculation. See <a href="#">fit.xyz</a> .
progress	progress bar for use with shiny web app.
...	other parameters for <a href="#">read.pdb</a> .

### Details

The input `aln`, produced with [read.fasta](#), must have identifiers (i.e. sequence names) that match the PDB file names. For example the sequence corresponding to the structure “1bg2.pdb” should have the identifier ‘1bg2’. See examples below.

Sequence miss-matches will generate errors. Thus, care should be taken to ensure that the sequences in the alignment match the sequences in their associated PDB files.

### Value

Returns a list of class “pdbs” with the following five components:

xyz	numeric matrix of aligned C-alpha coordinates.
resno	character matrix of aligned residue numbers.
b	numeric matrix of aligned B-factor values.
chain	character matrix of aligned chain identifiers.
id	character vector of PDB sequence/structure names.
ali	character matrix of aligned sequences.
resid	character matrix of aligned 3-letter residue names.
sse	character matrix of aligned helix and strand secondary structure elements as defined in each PDB file.
call	the matched call.

### Note

The sequence character ‘X’ is useful for masking unusual or unknown residues, as it can match any other residue type.

### Author(s)

Barry Grant

### References

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

### See Also

[read.fasta](#), [read.pdb](#), [core.find](#), [fit.xyz](#), [read.all](#), [pymol.pdbs](#)

## Examples

```
# Redundant testing excluded

# Read sequence alignment
file <- system.file("examples/kif1a.fa", package="bio3d")
aln <- read.fasta(file)

# Read aligned PDBs
pdbs <- read.fasta.pdb(aln)

# Structure/sequence names/ids
basename( pdbs$id )

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

# Alignment C-alpha coordinates for these positions
pdbs$xyz[, atom2xyz(335:339)]

# See 'fit.xyz()' function for actual coordinate superposition
# e.g. fit to first structure
# xyz <- fit.xyz(pdbs$xyz[1,], pdbs)
# xyz[, atom2xyz(335:339)]
```

---

read.mol2

*Read MOL2 File*

---

## Description

Read a Tripos MOL2 file

## Usage

```
read.mol2(file, maxlines = -1L)

## S3 method for class 'mol2'
print(x, ...)
```

## Arguments

**file** a single element character vector containing the name of the MOL2 file to be read.

maxlines	the maximum number of lines to read before giving up with large files. Default is all lines.
x	an object as obtained from read.mol2.
...	additional arguments to 'print'.

### Details

Basic functionality to parse a MOL2 file. The current version reads and stores '@<TRIPPO>MOLECULE', '@<TRIPPO>ATOM', '@<TRIPPO>BOND' and '@<TRIPPO>SUBSTRUCTURE' records.

In the case of a multi-molecule MOL2 file, each molecule will be stored as an individual 'mol2' object in a list. Conversely, if the multi-molecule MOL2 file contains identical molecules in different conformations (typically from a docking run), then the output will be one object with an atom and xyz component (xyz in matrix representation; row-wise coordinates).

See examples for further details.

### Value

Returns a list of molecules containing the following components:

atom	a data frame 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).
bond	a data frame containing all atomic bond information.
substructure	a data frame containing all substructure information.
xyz	a numeric matrix of ATOM coordinate data.
info	a numeric vector of MOL2 info data.
name	a single element character vector containing the molecule name.

### Note

For atom list components the column names can be used as a convenient means of data access, namely: Atom serial number "eleno", Atom name "elena", Orthogonal coordinates "x", Orthogonal coordinates "y", Orthogonal coordinates "z", Residue number "resno", Atom type "elety", Residue name "resid", Atom charge "charge", Status bit "statbit",

For bond list components the column names are: Bond identifier "id", number of the atom at one end of the bond "origin", number of the atom at the other end of the bond "target", the SYBYL bond type "type".

For substructure list components the column names are: substructure identifier "id", substructure name "name", the ID number of the substructure's root atom "root\_atom", the substructure type "subst\_type", the type of dictionary associated with the substructure "dict\_type", the chain to which the substructure belongs "chain", the subtype of the chain "sub\_type", the number of inter bonds "inter\_bonds", status bit "status".

See examples for further details.

### Author(s)

Lars Skjaerven

## References

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

## See Also

[write.mol2](#), [atom.select.mol2](#), [trim.mol2](#), [as.pdb.mol2](#) [read.pdb](#)

## Examples

```
cat("\n")
## Not run:
## Read a single entry MOL2 file
## (returns a single object)
mol <- read.mol2( system.file("examples/aspirin.mol2", package="bio3d") )

## Short summary of the molecule
print(mol)

## ATOM records
mol$atom

## BOND records
mol$bond

## Print some coordinate data
head(mol$atom[, c("x","y","z")])

## Or coordinates as a numeric vector
#head(mol$xyz)

## Print atom charges
head(mol$atom[, "charge"])

## Convert to PDB
pdb <- as.pdb(mol)

## Read a multi-molecule MOL2 file
## (returns a list of objects)
#multi.mol <- read.mol2("zinc.mol2")

## Number of molecules described in file
#length(multi.mol)

## Access ATOM records for the first molecule
#multi.mol[[1]]$atom

## Or coordinates for the second molecule
#multi.mol[[2]]$xyz
```

```

## Process output from docking (e.g. DOCK)
## (typically one molecule with many conformations)
## (returns one object, but xyz in matrix format)
#dock <- read.mol2("dock.mol2")

## Reference PDB file (e.g. X-ray structure)
#pdb <- read.pdb("dock_ref.pdb")

## Calculate RMSD of docking modes
#sele <- atom.select(dock, "noh")
#rmsd(pdb$xyz, dock$xyz, b.ind=sele$xyz)

## End(Not run)

```

---

read.ncdf

*Read AMBER Binary netCDF files*


---

## Description

Read coordinate data from a binary netCDF trajectory file.

## Usage

```

read.ncdf(trjfile, headonly = FALSE, verbose = TRUE, time = FALSE,
          first = NULL, last = NULL, stride = 1, cell = FALSE,
          at.sel = NULL)

```

## Arguments

trjfile	name of trajectory file to read. A vector if treat a batch of files
headonly	logical, if TRUE only trajectory header information is returned. If FALSE only trajectory coordinate data is returned.
verbose	logical, if TRUE print details of the reading process.
time	logical, if TRUE the first and last have the time unit ps; Otherwise the unit is the frame number.
first	starting time or frame number to read; If NULL, start from the beginning of the file(s).
last	read data until last time or frame number; If NULL or equal to -1, read until the end of the file(s).
stride	take at every stride frame(s)
cell	logical, if TRUE and headonly is FALSE return cell information only. Otherwise, return header or coordinates.
at.sel	an object of class 'select' indicating a subset of atomic coordinates to be read.

## Details

Reads a AMBER netCDF format trajectory file with the help of David W. Pierce's (UCSD) `ncdf4` package available from CRAN.

## Value

A list of trajectory header data, a numeric matrix of xyz coordinates with a frame/structure per row and a Cartesian coordinate per column, or a numeric matrix of cell information with a frame/structure per row and lengths and angles per column. If `time=TRUE`, row names of returned coordinates or cell are set to be the physical time of corresponding frames.

## Note

See AMBER documentation for netCDF format description.

NetCDF binary trajectory files are supported by the AMBER modules `sander`, `pmemd` and `ptraj`. Compared to formatted trajectory files, the binary trajectory files are smaller, higher precision and significantly faster to read and write.

NetCDF provides for file portability across architectures, allows for backwards compatible extensibility of the format and enables the files to be self-describing. Support for this format is available in VMD.

If you experience problems reading your trajectory file with `read.ncdf()` consider first reading your file into VMD and from there exporting a new DCD trajectory file with the 'save coordinates' option. This new file should be easily read with `read.dcd()`.

## Author(s)

Barry Grant

## References

Grant, B.J. et al. (2006) *Bioinformatics* **22**, 2695–2696. <https://www.unidata.ucar.edu/software/netcdf/> <https://cirrus.ucsd.edu/~pierce/ncdf/> <https://ambermd.org/FileFormats.php#netcdf>

## See Also

[read.dcd](#), [write.ncdf](#), [read.pdb](#), [write.pdb](#), [atom.select](#)

## Examples

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

## Write to netCDF format
write.ncdf(trj, "newtrj.nc")

## Read trj
```



```
trj <- read.ncdf("newtrj.nc")  
  
## End(Not run)
```

---

read.pdb

*Read PDB File*

---

## Description

Read a Protein Data Bank (PDB) coordinate file.

## Usage

```
read.pdb(file, maxlines = -1, multi = FALSE, rm.insert = FALSE,  
         rm.alt = TRUE, ATOM.only = FALSE, hex = FALSE, verbose = TRUE)
```

```
read.pdb2(file, maxlines = -1, multi = FALSE, rm.insert = FALSE,  
         rm.alt = TRUE, ATOM.only = FALSE, verbose = TRUE)
```

```
## S3 method for class 'pdb'  
print(x, printseq=TRUE, ...)
```

```
## S3 method for class 'pdb'  
summary(object, printseq=FALSE, ...)
```

## Arguments

file	a single element character vector containing the name of the PDB file to be read, or the four letter PDB identifier for online file access.
maxlines	the maximum number of lines to read before giving up with large files. By default it will read up to the end of input on the connection.
multi	logical, if TRUE multiple ATOM records are read for all models in multi-model files and their coordinates returned.
rm.insert	logical, if TRUE PDB insert records are ignored.
rm.alt	logical, if TRUE PDB alternate records are ignored.
ATOM.only	logical, if TRUE only ATOM/HETATM records are stored. Useful for speed enhancements with large files where secondary structure, biological unit and other remark records are not required.
hex	logical, if TRUE enable parsing of hexadecimal atom numbers (> 99.999) and residue numbers (> 9.999) (e.g. from VMD). Note that numbering is assumed to be consecutive (with no missing numbers) and the hexadecimals should start at atom number 100.000 and residue number 10.000 and proceed to the end of file.
verbose	print details of the reading process.
x	a PDB structure object obtained from <a href="#">read.pdb</a> .

object	a PDB structure object obtained from <a href="#">read.pdb</a> .
printseq	logical, if TRUE the PDB ATOM sequence will be printed to the screen. See also <a href="#">pdbseq</a> .
...	additional arguments to 'print'.

### Details

`read.pdb` is a re-implementation (using Rcpp) of the slower but more tested R implementation of the same function (called `read.pdb2` since `bio3d-v2.3`).

`maxlines` may be set so as to restrict the reading to a portion of input files. Note that the preferred means of reading large multi-model files is via binary DCD or NetCDF format trajectory files (see the [read.dcd](#) and [read.ncdf](#) functions).

### Value

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

atom	a data.frame containing all atomic coordinate ATOM and HETATM data, with a row per ATOM/HETATM and a column per record type. See below for details of the record type naming convention (useful for accessing columns).
helix	'start', 'end' and 'length' of H type sse, where start and end are residue numbers "resno".
sheet	'start', 'end' and 'length' of E type sse, where start and end are residue numbers "resno".
seqres	sequence from SEQRES field.
xyz	a numeric matrix of class "xyz" containing the ATOM and HETATM coordinate data.
calpha	logical vector with length equal to <code>nrow(atom)</code> with TRUE values indicating a C-alpha "elety".
remark	a list object containing information taken from 'REMARK' records of a "pdb". It can be used for building biological units (See <a href="#">biounit</a> ).
call	the matched call.

### Note

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

### Author(s)

Barry Grant

## References

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

For a description of PDB format (version3.3) see:

<http://www.wwpdb.org/documentation/format33/v3.3.html>.

## See Also

[atom.select](#), [write.pdb](#), [trim.pdb](#), [cat.pdb](#), [read.prmtop](#), [as.pdb](#), [read.dcd](#), [read.ncdf](#),  
[read.fasta.pdb](#), [read.fasta](#), [biunit](#)

## Examples

```
## Read a PDB file from the RCSB online database
#pdb <- read.pdb("4q21")

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

## Print a brief composition summary
pdb

## Examine the storage format (or internal *str*ucture)
str(pdb)

## Print data for the first four atom
pdb$atom[1:4,]

## Print some coordinate data
head(pdb$atom[, c("x","y","z")])

## Or coordinates as a numeric vector
#head(pdb$xyz)

## Print C-alpha coordinates (can also use 'atom.select' function)
head(pdb$atom[pdb$calpha, c("resid","eley","x","y","z")])
inds <- atom.select(pdb, eley="CA")
head( pdb$atom[inds$atom, ] )

## The atom.select() function returns 'indices' (row numbers)
## that can be used for accessing subsets of PDB objects, e.g.
inds <- atom.select(pdb,"ligand")
pdb$atom[inds$atom,]
pdb$xyz[inds$xyz]

## See the help page for atom.select() function for more details.

## Not run:
## Print SSE data for helix and sheet,
## see also dssp() and stride() functions
print.sse(pdb)
```

```

pdb$helix
pdb$sheet$start

## Print SEQRES data
pdb$seqres

## SEQRES as one letter code
aa321(pdb$seqres)

## Where is the P-loop motif in the ATOM sequence
inds.seq <- motif.find("G...GKT", pdbseq(pdb))
pdbseq(pdb)[inds.seq]

## Where is it in the structure
inds.pdb <- atom.select(pdb, resno=inds.seq, elty="CA")
pdb$atom[inds.pdb$atom,]
pdb$xyz[inds.pdb$xyz]

## View in interactive 3D mode
#view(pdb)

## End(Not run)

```

---

read.pdcBD

*Read PQR output from pdcBD File*


---

## Description

Read a pdcBD PQR coordinate file.

## Usage

```
read.pdcBD(file, maxlines = 50000, multi = FALSE, rm.insert = FALSE,
           rm.alt = TRUE, verbose = TRUE)
```

## Arguments

file	the name of the pdcBD PQR file to be read.
maxlines	the maximum number of lines to read before giving up with large files. Default is 50,000 lines.
multi	logical, if TRUE multiple ATOM records are read for all models in multi-model files.
rm.insert	logical, if TRUE PDB insert records are ignored.
rm.alt	logical, if TRUE PDB alternate records are ignored.
verbose	print details of the reading process.

## Details

maxlines may require increasing for some large multi-model files. The preferred means of reading such data is via binary DCD format trajectory files (see the [read.dcd](#) function).

## Value

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

atom	a character matrix containing all atomic coordinate ATOM data, with a row per ATOM and a column per record type. See below for details of the record type naming convention (useful for accessing columns).
het	a character matrix containing atomic coordinate records for atoms within "non-standard" HET groups (see atom).
helix	'start', 'end' and 'length' of H type sse, where start and end are residue numbers "resno".
sheet	'start', 'end' and 'length' of E type sse, where start and end are residue numbers "resno".
seqres	sequence from SEQRES field.
xyz	a numeric vector of ATOM coordinate data.
calpha	logical vector with length equal to nrow(atom) with TRUE values indicating a C-alpha "elety".

## Note

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

## Author(s)

Barry Grant

## References

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

For a description of PDB format (version3.3) see:

<http://www.wwpdb.org/documentation/format33/v3.3.html>.

## See Also

[atom.select](#), [write.pdb](#), [read.dcd](#), [read.fasta.pdb](#), [read.fasta](#)

**Examples**

```

# PDB server connection required - testing excluded

# Read a PDB file
pdb <- read.pdb( "1bg2" )

# Print data for the first atom
pdb$atom[1,]
# Look at the first het atom
pdb$het[1,]
# Print some coordinate data
pdb$atom[1:20, c("x","y","z")]

# Print C-alpha coordinates (can also use 'atom.select')
##pdb$xyz[pdb$calpha, c("resid","x","y","z")]

# Print SSE data (for helix and sheet)
pdb$helix
pdb$sheet$start

# Print SEQRES data
pdb$seqres

# Renumber residues
nums <- as.numeric(pdb$atom[, "resno"])
pdb$atom[, "resno"] <- nums - (nums[1] - 1)

# Write out renumbered PDB file
#write.pdb(pdb=pdb, file="eg.pdb")

```

---

read.pqr

*Read PQR File*


---

**Description**

Read a PQR coordinate file.

**Usage**

```

read.pqr(file, maxlines = -1, multi = FALSE, rm.insert = FALSE,
         rm.alt = TRUE, verbose = TRUE)

```

**Arguments**

file	the name of the PQR file to be read.
maxlines	the maximum number of lines to read before giving up with large files. By default it will read up to the end of input on the connection.

multi	logical, if TRUE multiple ATOM records are read for all models in multi-model files.
rm.insert	logical, if TRUE PDB insert records are ignored.
rm.alt	logical, if TRUE PDB alternate records are ignored.
verbose	print details of the reading process.

### Details

PQR file format is basically the same as PDB format except for the fields of o and b. In PDB, these two fields are filled with ‘Occupancy’ and ‘B-factor’ values, respectively, with each field 6-column long. In PQR, they are atomic ‘partial charge’ and ‘radii’ values, respectively, with each field 8-column long.

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 data.frame containing all atomic coordinate ATOM and HETATM data, with a row per ATOM/HETATM and a column per record type. See below for details of the record type naming convention (useful for accessing columns).
helix	‘start’, ‘end’ and ‘length’ of H type sse, where start and end are residue numbers “resno”.
sheet	‘start’, ‘end’ and ‘length’ of E type sse, where start and end are residue numbers “resno”.
seqres	sequence from SEQRES field.
xyz	a numeric matrix of class "xyz" containing the ATOM and HETATM coordinate data.
calpha	logical vector with length equal to nrow(atom) with TRUE values indicating a C-alpha “eley”.
call	the matched call.

### Note

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

### Author(s)

Barry Grant

## References

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

For a description of PDB format (version3.3) see:

<http://www.wwpdb.org/documentation/format33/v3.3.html>.

## See Also

[atom.select](#), [write.pqr](#), [read.pdb](#), [write.pdb](#), [read.dcd](#), [read.fasta.pdb](#), [read.fasta](#)

## Examples

```
# PDB server connection required - testing excluded

# Read a PDB file and write it as a PQR file
pdb <- read.pdb( "4q21" )
outfile = file.path(tempdir(), "eg.pqr")
write.pqr(pdb=pdb, file = outfile)

# Read the PQR file
pqr <- read.pqr(outfile)

## Print a brief composition summary
pqr

## Examine the storage format (or internal *str*ucture)
str(pqr)

## Print data for the first four atom
pqr$atom[1:4,]

## Print some coordinate data
head(pqr$atom[, c("x","y","z")])

## Print C-alpha coordinates (can also use 'atom.select' function)
head(pqr$atom[pqr$calpha, c("resid","eley","x","y","z")])
inds <- atom.select(pqr, eley="CA")
head( pqr$atom[inds$atom, ] )

## The atom.select() function returns 'indices' (row numbers)
## that can be used for accessing subsets of PDB objects, e.g.
inds <- atom.select(pqr,"ligand")
pqr$atom[inds$atom,]
pqr$xyz[inds$xyz]

## See the help page for atom.select() function for more details.
```



---

read.prmtop	<i>Read AMBER Parameter/Topology files</i>
-------------	--

---

### Description

Read parameter and topology data from an AMBER PrmTop file.

### Usage

```
read.prmtop(file)

## S3 method for class 'prmtop'
print(x, printseq=TRUE, ...)
```

### Arguments

file	a single element character vector containing the name of the PRMTOP file to be read.
x	a PRMTOP structure object obtained from <a href="#">read.prmtop</a> .
printseq	logical, if TRUE the residue sequence will be printed to the screen. See also <a href="#">pdbseq</a> .
...	additional arguments to 'print'.

### Details

This function provides basic functionality to read and parse a AMBER PrmTop file. The resulting 'prmtop' object contains a complete list object of the information stored in the PrmTop file.

See examples for further details.

### Value

Returns a list of class 'prmtop' (inherits class 'amber') with components according to the flags present in the PrmTop file. See the AMBER documentation for a complete list of flags/components: <http://ambermd.org/FileFormats.php>.

Selected components:

ATOM_NAME	a character vector of atom names.
ATOMS_PER_MOLECULE	a numeric vector containing the number of atoms per molecule.
MASS	a numeric vector of atomic masses.
RESIDUE_LABEL	a character vector of residue labels.
RESIDUE_RESIDUE_POINTER	a numeric vector of pointers to the first atom in each residue.
call	the matched call.

**Note**

See AMBER documentation for PrmTop format description:  
<http://ambermd.org/FileFormats.php>.

**Author(s)**

Lars Skjaerven

**References**

Grant, B.J. et al. (2006) *Bioinformatics* **22**, 2695–2696. <http://ambermd.org/FileFormats.php>

**See Also**

[read.crd](#), [read.ncdf](#), [as.pdb](#), [atom.select](#), [read.pdb](#)

**Examples**

```
## Not run:
## Read a PRMTOP file
prmtop <- read.prmtop(system.file("examples/crambin.prmtop", package="bio3d"))
print(prmtop)

## Explore prmtop file
head(prmtop$MASS)
head(prmtop$ATOM_NAME)

## Read Amber coordinates
crds <- read.crd(system.file("examples/crambin.inpcrd", package="bio3d"))

## Atom selection
ca.inds <- atom.select(prmtop, "calpha")

## Convert to PDB format
pdb <- as.pdb(prmtop, crds)
pdb.ca <- as.pdb(prmtop, crds, inds=ca.inds)

## Trajectory processing
#trj <- read.ncdf("traj.nc", at.sel=ca.inds)

## Convert to multimodel PDB format
#pdb <- as.pdb(prmtop, trj[1:20,], inds=ca.inds, inds.crd=NULL)

## RMSD of trajectory
#rd <- rmsd(crds$xyz[ca.inds$xyz], traj, fit=TRUE)

## End(Not run)
```

---

rgyr

*Radius of Gyration*

---

### Description

Calculate the radius of gyration of coordinate sets.

### Usage

```
rgyr(xyz, mass=NULL, ncore=1, nseg.scale=1)
```

### Arguments

xyz	a numeric vector, matrix or list object with an xyz component, containing one or more coordinate sets.
mass	a numeric vector of atomic masses (unit a.m.u.), or a PDB object with masses stored in the "B-factor" column. If mass=NULL, all atoms are assumed carbon.
ncore	number of CPU cores used to do the calculation. ncore>1 requires package 'parallel' installed.
nseg.scale	split input data into specified number of segments prior to running multiple core calculation. See <a href="#">fit.xyz</a> .

### Details

Radius of gyration is a standard measure of overall structural change of macromolecules.

### Value

Returns a numeric vector of radius of gyration.

### Author(s)

Xin-Qiu Yao & Pete Kekenos-Huskey

### References

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

### See Also

[fit.xyz](#), [rmsd](#), [read.pdb](#), [read.fasta.pdb](#)

**Examples**

```
# PDB server connection required - testing excluded

# -- Calculate Rog of single structure
pdb <- read.pdb("1bg2")
mass <- rep(12, length(pdb$xyz)/3)
mass[substr(pdb$atom[,"eley"], 1, 1) == "N"] <- 14
mass[substr(pdb$atom[,"eley"], 1, 1) == "H"] <- 1
mass[substr(pdb$atom[,"eley"], 1, 1) == "O"] <- 16
mass[substr(pdb$atom[,"eley"], 1, 1) == "S"] <- 32

rgyr(pdb, mass)

## Not run:
# -- Calculate Rog of a trajectory
xyz <- read.dcd(system.file("examples/hivp.dcd", package="bio3d"))
rg <- rgyr(xyz)
rg[1:10]

## End(Not run)
```

---

rle2

*Run Length Encoding with Indices*


---

**Description**

Compute the lengths, values and indices of runs of equal values in a vector. This is a modified 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 <code>rle()</code> ; an object of class "rle" for <code>inverse.rle()</code> .
...	further arguments; ignored here.
digits	number of significant digits for printing, see <a href="#">print.default</a> .
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(x)
## 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.ind=NULL, b.ind=NULL, fit=FALSE, ncore=1, nseg.scale=1)
```

**Arguments**

`a`                    a numeric vector containing the reference coordinate set for comparison with the coordinates in `b`. Alternatively, if `b=NULL` then `a` can be a matrix or list object containing multiple coordinates for pairwise comparison.

`b`                    a numeric vector, matrix or list object with an xyz component, containing one or more coordinate sets to be compared with `a`.

`a.ind`                a vector of indices that selects the elements of `a` upon which the calculation should be based.

`b.ind`                a vector of indices that selects the elements of `b` upon which the calculation should be based.

`fit`                  logical, if TRUE coordinate superposition is performed prior to RMSD calculation.

ncore	number of CPU cores used to do the calculation. ncore>1 requires package 'parallel' installed.
nseg.scale	split input data into specified number of segments prior to running multiple core calculation. See <a href="#">fit.xyz</a> .

### 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

```
# Redundant testing excluded

# -- Calculate RMSD between two or more structures
aln <- read.fasta(system.file("examples/kif1a.fa", package="bio3d"))
pdbs <- read.fasta.pdb(aln)

# Gap positions
inds <- gap.inspect(pdbs$xyz)

# Superposition before pairwise RMSD
rmsd(pdbs$xyz, fit=TRUE)

# RMSD between structure 1 and structures 2 and 3
rmsd(a=pdbs$xyz[1,], b=pdbs$xyz[2:3,], a.inds=inds$f.inds, b.inds=inds$f.inds, fit=TRUE)

# RMSD between structure 1 and all structures in alignment
rmsd(a=pdbs$xyz[1,], b=pdbs, a.inds=inds$f.inds, b.inds=inds$f.inds, fit=TRUE)

# RMSD without superposition
rmsd(pdbs$xyz)
```

rmsf

*Atomic RMS Fluctuations***Description**

Calculate atomic root mean squared fluctuations.

**Usage**

```
rmsf(xyz, grpby=NULL, average=FALSE)
```

**Arguments**

xyz	numeric matrix of coordinates with each row corresponding to an individual conformer.
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). If provided a 'pdb' object, grouping is automatically set by amino acid residues.
average	logical, if TRUE averaged over atoms.

**Details**

RMSF is an often used measure of conformational variance. It is calculated by

$$f_i = \sqrt{\frac{1}{M-1} \sum_j \|r_i^j - r_i^0\|^2}$$

, where  $f_i$  is the RMSF value for the  $i$ th atom,  $M$  the total number of frames (total number of rows of xyz),  $r_i^j$  the positional vector of the  $i$ th atom in the  $j$ th frame, and  $r_i^0$  the mean position of  $i$ th atom.  $\|r\|$  denotes the Euclidean norm of the vector  $r$ .

**Value**

Returns a numeric vector of RMSF values. If average=TRUE a single numeric value representing the averaged RMSF value over all atoms will be returned.

**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**

```
attach(transducin)

# Ignore Gaps
gaps <- gap.inspect(pdb$ali)

r <- rmsf(pdb$xyz)
plot(r[gaps$f.inds], typ="h", ylab="RMSF (A)")

detach(transducin)

## Not run:

pdb <- read.pdb("1d1d", multi=TRUE)
xyz <- pdb$xyz

# superimpose trajectory
xyz <- fit.xyz(xyz[1, ], xyz)

# select mainchain atoms
sele <- atom.select(pdb, elety=c("CA", "C", "N", "O"))

# residue numbers to group by
resno <- pdb$atom$resno[sele$atom]

# mean rmsf value of mainchain atoms of each residue
r <- rmsf(xyz[, sele$xyz], grpby=resno)
plot.bio3d(r, resno=pdb, sse=pdb, ylab="RMSF (A)")

## End(Not run)
```

---

rmsip

*Root Mean Square Inner Product*

---

**Description**

Calculate the RMSIP between two mode subspaces.

**Usage**

```
rmsip(...)

## S3 method for class 'enma'
```



```

rmsip(enma, ncore=NULL, subset=10, ...)

## Default S3 method:
rmsip(modes.a, modes.b, subset=10,
      row.name="a", col.name="b", ...)

```

### Arguments

enma	an object of class "enma" obtained from function <code>nma.pdbs</code> .
ncore	number of CPU cores used to do the calculation. <code>ncore&gt;1</code> requires package 'parallel' installed.
subset	the number of modes to consider.
modes.a	an object of class "pca" or "nma" as obtained from functions <code>pca.xyz</code> or <code>nma</code> .
modes.b	an object of class "pca" or "nma" as obtained from functions <code>pca.xyz</code> or <code>nma</code> .
row.name	prefix name for the rows.
col.name	prefix name for the columns.
...	arguments passed to associated functions.

### Details

RMSIP is a measure for the similarity between two set of modes obtained from principal component or normal modes analysis.

### Value

Returns an `rmsip` object with the following components:

overlap	a numeric matrix containing pairwise (squared) dot products between the modes.
rmsip	a numeric RMSIP value.

For function `rmsip.enma` a numeric matrix containing all pairwise RMSIP values of the modes stored in the `enma` object.

### Author(s)

Lars Skjaerven

### References

Skjaerven, L. et al. (2014) *BMC Bioinformatics* **15**, 399. Grant, B.J. et al. (2006) *Bioinformatics* **22**, 2695–2696. Amadei, A. et al. (1999) *Proteins* **36**, 19–424.

### See Also

[pca](#), [nma](#), [overlap](#).

Other similarity measures: [sip](#), [covoverlap](#), [bhattacharyya](#).

## Examples

```
## Not run:
# Load data for HIV example
trj <- read.dcd(system.file("examples/hivp.dcd", package="bio3d"))
pdb <- read.pdb(system.file("examples/hivp.pdb", package="bio3d"))

# Do PCA on simulation data
xyz.md <- fit.xyz(pdb$xyz, trj, fixed.inds=1:ncol(trj))
pc.sim <- pca.xyz(xyz.md)

# NMA
modes <- nma(pdb)

# Calculate the RMSIP between the MD-PCs and the NMA-MODEs
r <- rmsip(modes, pc.sim, subset=10, row.name="NMA", col.name="PCA")

# Plot pairwise overlap values
plot(r, xlab="NMA", ylab="PCA")

## End(Not run)
```

---

sdENM

*Index for the sdENM ff*

---

## Description

A dictionary of spring force constants for the sdENM force field.

## Usage

```
data(sdENM)
```

## Format

An array of 27 matrices containing the spring force constants for the 'sdENM' force field (see Dehouck et al for more information). Each matrix in the array holds the force constants for all amino acid pairs for a specific distance range.

See examples for more details.

## Source

Dehouck Y. & Mikhailov A.S. (2013) *PLoS Comput Biol* **9**:e1003209.

## References

Skjaerven, L. et al. (2014) *BMC Bioinformatics* **15**, 399. Grant, B.J. et al. (2006) *Bioinformatics* **22**, 2695–2696. Dehouck Y. et al. (2013) *PLoS Comput Biol* **9**:e1003209.

**Examples**

```
## Load force constant data
data(sdENM)

## force constants for amino acids A, C, D, E, and F
## in distance range [4, 4.5)
sdENM[1:5, 1:5, 1]

## and distance range [4.5, 5)
sdENM[1:5, 1:5, 2]

## amino acid pair A-P, at distance 4.2
sdENM["A", "P", 1]

## Not run:
## for use in NMA
pdb <- read.pdb( system.file("examples/1hel.pdb", package="bio3d") )
modes <- nma(pdb, ff="sdenm")

## End(Not run)
```

seq2aln

*Add a Sequence to an Existing Alignment***Description**

Add one or more sequences to an existing multiple alignment that you wish to keep intact.

**Usage**

```
seq2aln(seq2add, aln, id = "seq", file = "aln.fa", ...)
```

**Arguments**

seq2add	an sequence character vector or an alignment list object with <code>id</code> and <code>ali</code> components, similar to that generated by <a href="#">read.fasta</a> and <a href="#">seqaln</a> .
aln	an alignment list object with <code>id</code> and <code>ali</code> components, similar to that generated by <a href="#">read.fasta</a> and <a href="#">seqaln</a> .
id	a vector of sequence names to serve as sequence identifiers.
file	name of 'FASTA' output file to which alignment should be written.
...	additional arguments passed to <a href="#">seqaln</a> .

**Details**

This function calls the 'MUSCLE' program, to perform a profile profile alignment, which **MUST BE INSTALLED** on your system and in the search path for executables.

**Value**

A list with two components:

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

**Note**

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

**Author(s)**

Barry Grant

**References**

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

‘MUSCLE’ is the work of Edgar: Edgar (2004) *Nuc. Acid. Res.* **32**, 1792–1797.

Full details of the ‘MUSCLE’ algorithm, along with download and installation instructions can be obtained from:

<http://www.drive5.com/muscle/>.

**See Also**

[seqaln](#), [read.fasta](#), [read.fasta.pdb](#), [seqbind](#)

**Examples**

```
## Not run:
aa.1 <- pdbseq( read.pdb("1bg2") )
aa.2 <- pdbseq( read.pdb("3dc4") )
aa.3 <- pdbseq( read.pdb("1mkj") )

aln <- seqaln( seqbind(aa.1,aa.2) )

seq2aln(aa.3, aln)

## End(Not run)
```

---

 seqaln *Sequence Alignment with MUSCLE*


---

**Description**

Create multiple alignments of amino acid or nucleotide sequences according to the method of Edgar.

**Usage**

```
seqaln(aln, id=NULL, profile=NULL, exefile="muscle", outfile="aln.fa",
       protein=TRUE, seqgroup=FALSE, refine=FALSE, extra.args="",
       verbose=FALSE, web.args = list(), ...)
```

**Arguments**

aln	a sequence character matrix, as obtained from <a href="#">seqbind</a> , or an alignment list object as obtained from <a href="#">read.fasta</a> .
id	a vector of sequence names to serve as sequence identifiers.
profile	a profile alignment of class 'fasta' (e.g. obtained from <a href="#">read.fasta</a> ). The alignment aln will be added to the profile.
exefile	file path to the 'MUSCLE' program on your system (i.e. how is 'MUSCLE' invoked). Alternatively, 'CLUSTALO' can be used. Also supported is using the 'msa' package from Bioconductor (need to install packages using <code>BiocManager::install()</code> ). To do so, simply set <code>exefile="msa"</code> .
outfile	name of 'FASTA' output file to which alignment should be written.
protein	logical, if TRUE the input sequences are assumed to be protein not DNA or RNA.
seqgroup	logical, if TRUE similar sequences are grouped together in the output.
refine	logical, if TRUE the input sequences are assumed to already be aligned, and only tree dependent refinement is performed.
extra.args	a single character string containing extra command line arguments for the alignment program.
verbose	logical, if TRUE 'MUSCLE' warning and error messages are printed.
web.args	a 'list' object containing arguments to perform online sequence alignment using EMBL-EBI Web Services. See below for details.
...	additional arguments passed to the function <code>msa::msaMuscle()</code> .

**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 local ‘MUSCLE’ can not be found, alignment can still be performed via online web services (see below) with limited features.

If you have a large number of input sequences (a few thousand), or they are very long, the default settings may be too slow for practical use. A good compromise between speed and accuracy is to run just the first two iterations of the ‘MUSCLE’ algorithm by setting the `extra.args` argument to “-maxiters 2”.

You can set ‘MUSCLE’ to improve an existing alignment by setting `refine` to `TRUE`.

To inspect the sequence clustering used by ‘MUSCLE’ to produce alignments, include “-tree2 tree.out” in the `extra.args` argument. You can then load the “tree.out” file with the ‘read.tree’ function from the ‘ape’ package.

‘CLUSTALO’ can be used as an alternative to ‘MUSCLE’ by specifying `exefile='clustalo'`. This might be useful e.g. when adding several sequences to a profile alignment.

If local ‘MUSCLE’ or ‘CLUSTALO’ program is unavailable, the alignment can be performed via the ‘msa’ package from the Bioconductor repository. To do so, set `exefile="msa"`. Note that both ‘msa’ and ‘Biostrings’ packages need to be installed properly using `BiocManager::install()`.

If the access to any method mentioned above fails, the function will attempt to perform alignment via the EMBL-EBI Web Services (See <https://www.ebi.ac.uk/>). In this case, the argument `web.args` cannot be empty and must contain at least user’s E-Mail address. Note that as stated by EBI, a fake email address may result in your jobs being killed and your IP, organisation or entire domain being black-listed (See FAQs on <https://www.ebi.ac.uk/>). Possible parameters to be passed via `web.args` include:

**email** a string containing a valid E-Mail address. Required.

**title** a string for the title of the job to be submitted to the remote server. Optional.

**timeout** integer specifying the number of seconds to wait for the response of the server before a time out occurs. Default: 90.

An example of usage is `web.args=list(email='user_id@email.provider')`.

## Value

Returns a list of class “fasta” with the following components:

<code>ali</code>	an alignment character matrix with a row per sequence and a column per equivalent aminoacid/nucleotide.
<code>id</code>	sequence names as identifiers.
<code>call</code>	the matched call.

## Note

A system call is made to the ‘MUSCLE’ program, which must be installed on your system and in the search path for executables. See [http://thegrantlab.org/bio3d/articles/online/install\\_vignette/Bio3D\\_install.html](http://thegrantlab.org/bio3d/articles/online/install_vignette/Bio3D_install.html) for instructions of how to install this program.

## Author(s)

Barry Grant

## References

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

‘MUSCLE’ is the work of Edgar: Edgar (2004) *Nuc. Acid. Res.* **32**, 1792–1797.

Full details of the ‘MUSCLE’ algorithm, along with download and installation instructions can be obtained from:

<http://www.drive5.com/muscle/>.

## See Also

[read.fasta](#), [read.fasta.pdb](#), [get.seq](#), [seqbind](#), [pdbaln](#), [plot.fasta](#), [blast.pdb](#)

## Examples

```
## Not run:
##-- Basic sequence alignment
seqs <- get.seq(c("4q21_A", "1ftn_A"))
aln <- seqaln(seqs)

##-- add a sequence to the (profile) alignment
seq <- get.seq("1tnd_A")
aln <- seqaln(seq, profile=aln)

##-- Read a folder/directory of PDB files
#pdb.path <- "my_dir_of_pdbs"
#files <- list.files(path=pdb.path ,
#                   pattern=".pdb",
#                   full.names=TRUE)

##-- Use online files
files <- get.pdb(c("4q21", "1ftn"), URLonly=TRUE)

##-- Extract and store sequences
raw <- NULL
for(i in 1:length(files)) {
  pdb <- read.pdb(files[i])
  raw <- seqbind(raw, pdbseq(pdb) )
}

##-- Align these sequences
aln <- seqaln(raw, id=files, outfile="seqaln.fa")

##-- Read Aligned PDBs storing coordinate data
pdbs <- read.fasta.pdb(aln)

## Sequence identity
seqidentity(aln)

## Note that all the above can be done with the pdbaln() function:
#pdbs <- pdbaln(files)
```

```
##-- For identical sequences with masking use a custom matrix
aa <- seqbind(c("X","C","X","X","A","G","K"),
             c("C","-","A","X","G","X","X","K"))

aln <- seqaln(aln=aln, id=c("a","b"), outfile="temp.fas", protein=TRUE,
            extra.args= paste("-matrix",
                              system.file("matrices/custom.mat", package="bio3d"),
                              "-gapopen -3.0 ",
                              "-gapextend -0.5",
                              "-center 0.0" )

## End(Not run)
```

---

seqaln.pair

*Sequence Alignment of Identical Protein Sequences*


---

## Description

Create multiple alignments of amino acid sequences according to the method of Edgar.

## Usage

```
seqaln.pair(aln, ...)
```

## Arguments

aln	a sequence character matrix, as obtained from <a href="#">seqbind</a> , or an alignment list object as obtained from <a href="#">read.fasta</a> .
...	additional arguments for the function <a href="#">seqaln</a> .

## 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 identifiers.



**Note**

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

**Author(s)**

Barry Grant

**References**

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

‘MUSCLE’ is the work of Edgar: Edgar (2004) *Nuc. Acid. Res.* **32**, 1792–1797.

Full details of the ‘MUSCLE’ algorithm, along with download and installation instructions can be obtained from:

<http://www.drive5.com/muscle/>.

**See Also**

[seqaln](#), [read.fasta](#), [read.fasta.pdb](#), [seqbind](#)

**Examples**

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

##- Aligning a PDB sequence to an existing sequence alignment

##- Simple example
aln <- seqbind(c("X", "C", "X", "X", "A", "G", "K"),
              c("C", "-", "A", "X", "G", "X", "X", "K"))

seqaln.pair(aln, outfile = tempfile())

}
```

---

seqbind

*Combine Sequences by Rows Without Recycling*

---

**Description**

Take vectors and/or matrices arguments and combine them row-wise without recycling them (as is the case with [rbind](#)).

**Usage**

```
seqbind(..., blank = "-")
```

**Arguments**

... vectors, matrices, and/or alignment 'fasta' objects to combine.

blank a character to add to short arguments, to achieve the same length as the longer argument.

**Value**

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

ali an alignment character matrix with a row per sequence and a column per equivalent aminoacid/nucleotide.

id sequence names as identifiers.

call the matched call.

**Author(s)**

Barry Grant

**References**

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

**See Also**

[seqaln](#), [read.fasta](#), [read.pdb](#), [write.fasta](#), [rbind](#)

**Examples**

```
## Not run:
## Read two pdbs
a.pdb <- read.pdb("1bg2")
b.pdb <- read.pdb("1goj")

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

# seqaln(seqs)

## End(Not run)
```

---

seqidentity	<i>Percent Identity</i>
-------------	-------------------------

---

### Description

Determine the percent identity scores for aligned sequences.

### Usage

```
seqidentity(alignment, normalize=TRUE, similarity=FALSE, ncore=1, nseg.scale=1)
```

### Arguments

alignment	sequence alignment obtained from <a href="#">read.fasta</a> or an alignment character matrix.
normalize	logical, if TRUE output is normalized to values between 0 and 1 otherwise percent identity is returned.
similarity	logical, if TRUE sequence similarity is calculated instead of identity.
ncore	number of CPU cores used to do the calculation. <code>ncore&gt;1</code> requires package ‘parallel’ installed.
nseg.scale	split input data into specified number of segments prior to running multiple core calculation. See <a href="#">fit.xyz</a> .

### 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](#), [filter.identity](#), [entropy](#), [consensus](#)

## Examples

```
attach(kinesin)

ide.mat <- seqidentity(pdb)

# Plot identity matrix
plot.dmat(ide.mat, color.palette=mono.colors,
          main="Sequence Identity", xlab="Structure No.",
          ylab="Structure No.")

# Histogram of pairwise identity values
hist(ide.mat[upper.tri(ide.mat)], breaks=30,xlim=c(0,1),
     main="Sequence Identity", xlab="Identity")

# Compare two sequences
seqidentity( rbind(pdb$ali[1,], pdb$ali[15,]) )

detach(kinesin)
```

---

setup.ncore

*Setup for Running Bio3D Functions using Multiple CPU Cores*

---

## Description

Internally used in parallelized Bio3D functions.

## Usage

```
setup.ncore(ncore, bigmem = FALSE)
```

## Arguments

ncore	User set (or default) value of 'ncore'.
bigmem	logical, if TRUE also check the availability of 'bigmemory' package.

## Details

Check packages and set correct value of 'ncore'.

## Value

The actual value of 'ncore'.

## Examples

```
setup.ncore(NULL)
setup.ncore(1)
# setup.ncore(2)
```

---

`sip`*Square Inner Product*

---

**Description**

Calculate the correlation between two atomic fluctuation vectors.

**Usage**

```
sip(...)  
  
## S3 method for class 'nma'  
sip(a, b, ...)  
  
## S3 method for class 'enma'  
sip(enma, ncore=NULL, ...)  
  
## Default S3 method:  
sip(v, w, ...)
```

**Arguments**

<code>enma</code>	an object of class "enma" obtained from function <code>nma.pdbs</code> .
<code>ncore</code>	number of CPU cores used to do the calculation. <code>ncore&gt;1</code> requires package 'parallel' installed.
<code>a</code>	an 'nma' object as object from function <code>nma</code> to be compared to <code>b</code> .
<code>b</code>	an 'nma' object as object from function <code>nma</code> to be compared to <code>a</code> .
<code>v</code>	a numeric vector containing the atomic fluctuation values.
<code>w</code>	a numeric vector containing the atomic fluctuation values.
<code>...</code>	arguments passed to associated functions.

**Details**

SIP is a measure for the similarity of atomic fluctuations of two proteins, e.g. experimental b-factors, theoretical RMSF values, or atomic fluctuations obtained from NMA.

**Value**

Returns the similarity coefficient(s).

**Author(s)**

Lars Skjaerven

## References

Skjaerven, L. et al. (2014) *BMC Bioinformatics* **15**, 399. Grant, B.J. et al. (2006) *Bioinformatics* **22**, 2695–2696. Fuglebakk, E. et al. (2013) *JCTC* **9**, 5618–5628.

## See Also

Other similarity measures: [covoverlap](#), [bhattacharyya](#), [rmsip](#).

## Examples

```
pdb <- read.pdb( system.file("examples/1hel.pdb", package="bio3d") )
a <- nma(pdb)
b <- nma(pdb, ff="anm")

sip(a$fluctuations, b$fluctuations)
```

---

sse.bridges

*SSE Backbone Hydrogen Bonding*

---

## Description

Determine backbone C=O to N-H hydrogen bonding in secondary structure elements.

## Usage

```
sse.bridges(sse, type="helix", hbond=TRUE, energy.cut=-1.0)
```

## Arguments

sse	an sse object as obtained with dssp.
type	character string specifying ‘helix’ or ‘sheet’.
hbond	use hbond records in the dssp output.
energy.cut	cutoff for the dssp hbond energy.

## Details

Simple functionality to parse the ‘BP’ and ‘hbond’ records of the DSSP output.  
Requires input from function dssp with arguments resno=FALSE and full=TRUE.

## Value

Returns a numeric matrix of two columns containing the residue ids of the paired residues.

## Author(s)

Lars Skjaerven

## References

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

## See Also

[read.pdb](#), [dssp](#)

## Examples

```
## Not run:
# Read a PDB file
pdb <- read.pdb( system.file("examples/1hel.pdb", package="bio3d") )
sse <- dssp(pdb, resno=FALSE, full=TRUE)

sse.bridges(sse, type="helix")

## End(Not run)
```

---

store.atom	<i>Store all-atom data from a PDB object</i>
------------	--

---

## Description

Not intended for public usage

## Usage

```
store.atom(pdb=NULL)
```

## 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. If pdb=NULL, returns the default atom names to be stored.

## Note

This function is still in development and is NOT part of the official bio3d package

## Author(s)

Barry Grant

**References**

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

**See Also**

[read.fasta.pdb](#)

**Examples**

```
## Not run:
pdb <- read.pdb( get.pdb("5p21", URLonly=TRUE) )
a <- store.atom(pdb)
a[,1:2]

## End(Not run)
```

---

struct.aln

*Structure Alignment Of Two PDB Files*

---

**Description**

Performs a sequence and structural alignment of two PDB entities.

**Usage**

```
struct.aln(fixed, mobile, fixed.indxs=NULL, mobile.indxs=NULL,
           write.pdbs=TRUE, outpath = "fitlsq", prefix=c("fixed",
           "mobile"), max.cycles=10, cutoff=0.5, ... )
```

**Arguments**

fixed	an object of class <code>pdb</code> as obtained from function <code>read.pdb</code> .
mobile	an object of class <code>pdb</code> as obtained from function <code>read.pdb</code> .
fixed.indxs	atom and xyz coordinate indices obtained from <code>atom.select</code> that selects the elements of <code>fixed</code> upon which the calculation should be based.
mobile.indxs	atom and xyz coordinate indices obtained from <code>atom.select</code> that selects the elements of <code>mobile</code> upon which the calculation should be based.
write.pdbs	logical, if <code>TRUE</code> the aligned structures are written to PDB files.
outpath	character string specifying the output directory when <code>write.pdbs</code> is <code>TRUE</code> .
prefix	a character vector of length 2 containing the filename prefix in which the fitted structures should be written.
max.cycles	maximum number of refinement cycles.
cutoff	standard deviation of the pairwise distances for aligned residues at which the fitting refinement stops.
...	extra arguments passed to <code>seqaln</code> function.



## Details

This function performs a sequence alignment followed by a structural alignment of the two PDB entities. Cycles of refinement steps of the structural alignment are performed to improve the fit by removing atoms with a high structural deviation. The primary purpose of the function is to allow rapid structural alignment (and RMSD analysis) for protein structures with unequal, but related sequences.

The function reports the residues of `fixed` and `mobile` included in the final structural alignment, as well as the related RMSD values.

This function makes use of the underlying functions `seqaln`, `rot.lsqr`, and `rmsd`.

## Value

Returns a list with the following components:

<code>a.inds</code>	atom and xyz indices of <code>fixed</code> .
<code>b.inds</code>	atom and xyz indices of <code>mobile</code> .
<code>xyz</code>	fitted xyz coordinates of <code>mobile</code> .
<code>rmsd</code>	a numeric vector of RMSD values after each cycle of refinement.

## Author(s)

Lars Skjarven

## References

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

## See Also

[rmsd](#), [rot.lsqr](#), [seqaln](#), [pdbaln](#)

## Examples

```
# Needs MUSCLE installed - testing excluded

if(check.utility("muscle")) {

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

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

  ## Align and fit b on to a:
  path = file.path(tempdir(), "struct.aln")
  aln <- struct.aln(a, b, outpost = path, outfile = tempfile())

  ## Should be the same as aln$rmsd (when using aln$a.inds and aln$b.inds)
```

```
rmsd(a$xyz, b$xyz, aln$a.inds$xyz, aln$b.inds$xyz, fit=TRUE)

invisible( cat("\nSee the output files:", list.files(path, full.names = TRUE), sep="\n" )
}

## Not run:
## Align two subunits of GroEL (open and closed states)
a <- read.pdb("1sx4")
b <- read.pdb("1xck")

## Select chain A only
a.inds <- atom.select(a, chain="A")
b.inds <- atom.select(b, chain="A")

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

## End(Not run)
```

---

torsion.pdb

*Calculate Mainchain and Sidechain Torsion/Dihedral Angles*

---

## Description

Calculate all torsion angles for a given protein PDB structure object.

## Usage

```
torsion.pdb(pdb)
```

## Arguments

pdb                    a PDB structure object as obtained from function read.pdb.

## Details

The conformation of a polypeptide chain can be usefully described in terms of angles of internal rotation around its constituent bonds. See the related `torsion.xyz` function, which is called by this function, for details.

## Value

Returns a list object with the following components:

phi	main chain torsion angle for atoms C,N,CA,C.
psi	main chain torsion angle for atoms N,CA,C,N.
omega	main chain torsion angle for atoms CA,C,N,CA.

alpha	virtual torsion angle between consecutive C-alpha atoms.
chi1	side chain torsion angle for atoms N,CA,CB,*G.
chi2	side chain torsion angle for atoms CA,CB,*G,*D.
chi3	side chain torsion angle for atoms CB,*G,*D,*E.
chi4	side chain torsion angle for atoms *G,*D,*E,*Z.
chi5	side chain torsion angle for atoms *D,*E,*Z, NH1.
coords	numeric matrix of 'justified' coordinates.
tbl	a numeric matrix of psi, phi and chi torsion angles.

### Note

For the protein backbone, or main-chain atoms, the partial double-bond character of the peptide bond between 'C=N' atoms severely restricts internal rotations. In contrast, internal rotations around the single bonds between 'N-CA' and 'CA-C' are only restricted by potential steric collisions. Thus, to a good approximation, the backbone conformation of each residue in a given polypeptide chain can be characterised by the two angles phi and psi.

Sidechain conformations can also be described by angles of internal rotation denoted chi1 up to chi5 moving out along the sidechain.

### Author(s)

Barry Grant

### References

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

### See Also

[torsion.xyz](#), [read.pdb](#), [dssp](#), [stride](#).

### Examples

```
# PDB server connection required - testing excluded

##-- PDB torsion analysis
pdb <- read.pdb( "1bg2" )
tor <- torsion.pdb(pdb)
head(tor$tbl)

## basic Ramachandran plot
plot(tor$phi, tor$psi)

## torsion analysis of a single coordinate vector
#inds <- atom.select(pdb,"calpha")
#tor.ca <- torsion.xyz(pdb$xyz[inds$xyz], atm.inc=1)
```

```
##-- Compare two PDBs to highlight interesting residues
aln <- read.fasta(system.file("examples/kif1a.fa",package="bio3d"))
m <- read.fasta.pdb(aln)
a <- torsion.xyz(m$xyz[1,],1)
b <- torsion.xyz(m$xyz[2,],1)
d <- wrap.tor(a-b)
plot(m$resno[1,],d, typ="h")
```

---

torsion.xyz

*Calculate Torsion/Dihedral Angles*

---

### Description

Defined from the Cartesian coordinates of four successive atoms (A-B-C-D) the torsion or dihedral angle is calculated about an axis defined by the middle pair of atoms (B-C).

### Usage

```
torsion.xyz(xyz, atm.inc = 4)
```

### Arguments

xyz	a numeric vector of Cartesian 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 successive 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 successive evaluations).

### Value

A numeric vector of torsion angles.

**Note**

Contributions from Barry Grant.

**Author(s)**

Karim ElSawy

**References**

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

**See Also**

[torsion.pdb](#), [pca.tor](#), [wrap.tor](#), [read.pdb](#), [read.dcd](#).

**Examples**

```
## Calculate torsions for cis & trans conformers
xyz <- rbind(c(0,-0.5,0,1,0,0,1,1,0,0,1.5,0),
            c(0,-0.5,0,1,0,0,1,1,0,2,1.5,0))-3)
cis.tor <- torsion.xyz( xyz[1,] )
trans.tor <- torsion.xyz( xyz[2,] )
apply(xyz, 1, torsion.xyz)

plot(range(xyz), range(xyz), xlab="", ylab="", typ="n", axes=FALSE)
  apply(xyz, 1, function(x){
    lines(matrix(x, ncol=3, byrow=TRUE), lwd=4)
    points(matrix(x, ncol=3, byrow=TRUE), cex=2.5,
            bg="white", col="black", pch=21) } )

  text( t(apply(xyz, 1, function(x){
    apply(matrix(x, ncol=3, byrow=TRUE)[c(2,3),], 2, mean) })),
        labels=c(0,180), adj=-0.5, col="red")

# PDB server connection required - testing excluded

##-- PDB torsion analysis
pdb <- read.pdb("1bg2")
tor <- torsion.pdb(pdb)
## basic Ramachandran plot
plot(tor$phi, tor$psi)

## torsion analysis of a single coordinate vector
inds <- atom.select(pdb,"calpha")
tor.ca <- torsion.xyz(pdb$xyz[inds$xyz], atm.inc=3)

##-- Compare two PDBs to highlight interesting residues
aln <- read.fasta(system.file("examples/kif1a.fa",package="bio3d"))
m <- read.fasta.pdb(aln)
```

```

a <- torsion.xyz(m$xyz[1,],1)
b <- torsion.xyz(m$xyz[2,],1)
## Note the periodicity of torsion angles
d <- wrap.tor(a-b)
plot(m$resno[1,],d, typ="h")

```

---

trim

*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(...)

## S3 method for class 'pdb'
trim(pdb, ..., inds = NULL, sse = TRUE)

```

### Arguments

pdb	a PDB structure object obtained from <a href="#">read.pdb</a> .
...	additional arguments passed to <a href="#">atom.select</a> . If inds is also provided, these arguments will be ignored.
inds	a list object of ATOM and XYZ indices as obtained from <a href="#">atom.select</a> . If NULL, atom selection will be obtained from calling <code>atom.select(pdb, ...)</code> .
sse	logical, if 'FALSE' helix and sheet components are omitted from output.

### Details

This is a basic utility function for creating a new PDB object based on a selection of atoms.

### Value

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

atom	a character matrix containing all atomic coordinate ATOM data, with a row per ATOM and a column per record type. See below for details of the record type naming convention (useful for accessing columns).
het	a character matrix containing atomic coordinate records for atoms within "non-standard" HET groups (see atom).
helix	'start', 'end' and 'length' of H type sse, where start and end are residue numbers "resno".

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

### Note

het and seqres list components are returned unmodified.

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

### Author(s)

Barry Grant, Lars Skjaerven

### References

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

For a description of PDB format (version3.3) see:

<http://www.wwpdb.org/documentation/format33/v3.3.html>.

### See Also

[trim.pdbs](#), [trim.xyz](#), [read.pdb](#), [atom.select](#)

### Examples

```
## Not run:
## Read a PDB file from the RCSB online database
pdb <- read.pdb("1bg2")

## Select calpha atoms
sele <- atom.select(pdb, "calpha")

## Trim PDB
new.pdb <- trim.pdb(pdb, inds=sele)

## Or, simply
#new.pdb <- trim.pdb(pdb, "calpha")

## Write to file
write.pdb(new.pdb, file="calpha.pdb")

## End(Not run)
```

---

`trim.mol2`*Trim a MOL2 Object To A Subset of Atoms.*

---

### Description

Produce a new smaller MOL2 object, containing a subset of atoms, from a given larger MOL2 object.

### Usage

```
## S3 method for class 'mol2'  
trim(mol, ..., inds = NULL)
```

### Arguments

<code>mol</code>	a MOL2 structure object obtained from <a href="#">read.mol2</a> .
<code>...</code>	additional arguments passed to <a href="#">atom.select</a> . If <code>inds</code> is also provided, these arguments will be ignored.
<code>inds</code>	a list object of ATOM and XYZ indices as obtained from <a href="#">atom.select</a> . If NULL, atom selection will be obtained from calling <code>atom.select(mol, ...)</code> .

### Details

This is a basic utility function for creating a new MOL2 object based on a selection of atoms.

### Value

Returns a list of class "mol2".

### Author(s)

Lars Skjaerven

### References

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

### See Also

[read.mol2](#), [atom.select.mol2](#), [as.pdb.mol2](#), [write.mol2](#),



**Examples**

```
## Not run:
## Read a MOL2 file from those included with the package
mol <- read.mol2( system.file("examples/aspirin.mol2", package="bio3d"))

## Trim away H-atoms
mol <- trim(mol, "noh")

## End(Not run)
```

---

trim.pdbs	<i>Filter or Trim a PDBs Object</i>
-----------	-------------------------------------

---

**Description**

Trim residues and/or filter out structures from a PDBs object.

**Usage**

```
## S3 method for class 'pdbs'
trim(pdbs, row.inds=NULL, col.inds=NULL, ...)
```

**Arguments**

pdbs	an object of class pdbs as obtained from function pdbaln or read.fasta.pdb; a xyz matrix containing the cartesian coordinates of C-alpha atoms.
row.inds	a numeric vector of indices pointing to the PDB structures to keep (rows in the pdbs\$ali matrix).
col.inds	a numeric vector of indices pointing to the alignment columns to keep (columns in the pdbs\$ali matrix).
...	additional arguments passed to and from functions.

**Details**

Utility function to remove structures, or trim off columns, in a 'pdbs' object.

**Value**

Returns an updated 'pdbs' object with the following components:

xyz	numeric matrix of aligned C-alpha coordinates.
resno	character matrix of aligned residue numbers.
b	numeric matrix of aligned B-factor values.
chain	character matrix of aligned chain identifiers.
id	character vector of PDB sequence/structure names.
ali	character matrix of aligned sequences.
call	the matched call.

**Author(s)**

Lars Skjaerven

**References**

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

**See Also**

[pdbaln](#), [gap.inspect](#), [read.fasta](#), [read.fasta.pdb](#), [trim.pdb](#),

**Examples**

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

## Sequence Alignment, and connectivity check
pdbs <- pdbaln(files)

cons <- inspect.connectivity(pdbs)

## omit files with missing residues
trim.pdb(pdb, row.ind=which(cons))

## End(Not run)
```

---

trim.xyz

*Trim a XYZ Object of Cartesian Coordinates.*

---

**Description**

Produce a new smaller XYZ object, containing a subset of atoms.

**Usage**

```
## S3 method for class 'xyz'
trim(xyz, row.ind = NULL, col.ind = NULL, ...)
```

**Arguments**

xyz	a XYZ object containing Cartesian coordinates, e.g. obtained from <a href="#">read.pdb</a> , <a href="#">read.ncdf</a> .
row.ind	a numeric vector specifying which rows of the xyz matrix to return.
col.ind	a numeric vector specifying which columns of the xyz matrix to return.
...	additional arguments passed to and from functions.

**Details**

This function provides basic functionality for subsetting a matrix of class 'xyz' while also maintaining the class attribute.

**Value**

Returns an object of class xyz with the Cartesian coordinates stored in a matrix object with dimensions  $M \times 3N$ , where  $N$  is the number of atoms, and  $M$  number of frames.

**Author(s)**

Lars Skjaerven

**References**

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

**See Also**

[read.pdb](#), [as.xyz](#).

**Examples**

```
## Not run:
## Read a PDB file from the RCSB online database
pdb <- read.pdb("1bg2")

## Select calpha atoms
sele <- atom.select(pdb, "calpha")

## Trim XYZ
trim(pdb$xyz, col.inds=sele$xyz)

## Equals to
pdb$xyz[, sele$xyz, drop=FALSE]

## End(Not run)
```

---

unbound

*Sequence Generation from a Bounds Vector*

---

**Description**

Generate a sequence of consecutive numbers from a [bounds](#) vector.

**Usage**

```
unbound(start, end = NULL)
```

**Arguments**

start	vector of starting values, or a matrix containing starting and end values such as that obtained from <a href="#">bounds</a> .
end	vector of (maximal) end values, such as that obtained from <a href="#">bounds</a> .

**Details**

This is a simple utility function that does the opposite of the [bounds](#) function. If `start` is a vector, `end` must be a vector having the same length as `start`. If `start` is a matrix with column names contain 'start' and 'end', such as that returned from [bounds](#), `end` can be skipped and both starting and end values will be extracted from `start`.

**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)
```

---

uniprot

*Fetch UniProt Entry Data.*

---

**Description**

Fetch protein sequence and functional information from the UniProt database.

**Usage**

```
uniprot(accid)
```

**Arguments**

accid            UniProt accession id.

**Details**

This is a basic utility function for downloading information from the UniProt database. UniProt contains protein sequence and functional information.

**Value**

Returns a list object with the following components:

accession	a character vector with UniProt accession id's.
name	abbreviated name.
fullName	full recommended protein name.
shortName	short protein name.
sequence	protein sequence.
gene	gene names.
organism	organism.
taxon	taxonomic lineage.

**Author(s)**

Lars Skjaerven

**References**

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

See also the UniProt web-site for more information:

<https://www.uniprot.org/>.

**See Also**

[blast.pdb](#), [get.seq](#)

**Examples**

```
## Not run:  
# UNIPROT server connection required - testing excluded  
  
prot <- uniprot('PH4H_HUMAN')  
prot$fullName  
prot$sequence  
  
## End(Not run)
```

---

`var.xyz`*Pairwise Distance Variance in Cartesian Coordinates*

---

**Description**

Calculate the variance of all pairwise distances in an ensemble of Cartesian coordinates.

**Usage**

```
var.xyz(xyz, weights=TRUE)
var.pdbs(pdbs, ...)
```

**Arguments**

<code>xyz</code>	an object of class "xyz" containing Cartesian coordinates in a matrix.
<code>weights</code>	logical, if TRUE weights are calculated based on the pairwise distance variance.
<code>pdbs</code>	a 'pdbs' object as object from function <code>pdba1n</code> .
<code>...</code>	arguments passed to associated functions.

**Details**

This function calculates the variance of all pairwise distances in an ensemble of Cartesian coordinates. The primary use of this function is to calculate weights to scale the pair force constant for NMA.

**Value**

Returns the a matrix of the pairwise distance variance, formatted as weights if 'weights=TRUE'.

**Author(s)**

Lars Skjaerven

**References**

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

**See Also**

[nma.pdbs](#)

---

`vec2resno`*Replicate Per-residue Vector Values*

---

### Description

Replicate values in one vector based on consecutive entries in a second vector. Useful for adding per-residue data to all-atom PDB files.

### Usage

```
vec2resno(vec, resno)
```

### Arguments

<code>vec</code>	a vector of values to be replicated.
<code>resno</code>	a reference vector or a PDB structure object, obtained from <a href="#">read.pdb</a> , 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))
```

vmd

*View CNA Protein Structure Network Community Output in VMD***Description**

This function generates a VMD scene file and a PDB file that can be read and rendered by the VMD molecular viewer. Chose ‘color by chain’ to see corresponding regions of structure colored by community along with the community protein structure network.

**Usage**

```
vmd(...)

## S3 method for class 'cna'
vmd(x, pdb, layout = layout.cna(x, pdb, k=3),
    col.sphere=NULL, col.lines = "silver", weights = NULL,
    radius = table(x$communities$membership)/5, alpha = 1,
    vmdfile = "network.vmd", pdbfile = "network.pdb",
        full = FALSE, launch = FALSE, exefile=NULL, ...)
## S3 method for class 'ecna'
vmd(x, n=1, ...)
## S3 method for class 'cnapath'
vmd(x, pdb, out.prefix = "vmd.cnapath", spline = FALSE,
    colors = c("blue", "red"), launch = FALSE, exefile=NULL, mag=1.0, ...)
## S3 method for class 'ecnapath'
vmd(x, ...)
```

**Arguments**

x	A ‘cna’ or ‘cnapath’ class object, or a list of such objects, as obtained from functions <code>cna</code> or <code>cnapath</code> .
n	The index to indicate which network or path to view with <code>vmd</code> .
pdb	A ‘pdb’ class object such as obtained from ‘ <code>read.pdb</code> ’ function.
layout	A numeric matrix of Nx3 XYZ coordinate matrix, where N is the number of community spheres to be drawn.
col.sphere	A numeric vector containing the sphere colors.
col.lines	A character object specifying the color of the edges (default ‘silver’). Must use VMD colors names.
weights	A numeric vector specifying the edge width. Default is taken from <code>E(x\$community.network)\$weight</code> .
radius	A numeric vector containing the sphere radii. Default is taken from the number of community members divided by 5.
alpha	A single element numeric vector specifying the VMD alpha transparency parameter. Default is set to 1.
vmdfile	A character element specifying the output VMD scene file name that will be loaded in VMD.



pdbfile	A character element specifying the output pdb file name to be loaded in VMD.
full	Logical, if TRUE the full all-atom network rather than the clustered community network will be drawn. Intra community edges are colored according to the community membership, while inter community edges are thicker and colored black.
launch	Logical. If TRUE, a VMD session will be started with the output of 'vmd.cna'.
out.prefix	Prefix for the names of output files, 'vmd.cnapath.vmd' and 'vmd.cnapath.pdb'.
spline	Logical, if TRUE all paths are displayed as spline curves.
colors	Character vector or integer scalar, define path colors. If a character vector, passed to <code>colorRamp</code> function to generate the color scales. If an integer, color all paths the same way with VMD color ID equal to the integer.
exefile	file path to the 'VMD' program on your system (i.e. how is 'VMD' invoked). If NULL, use OS-dependent default path to the program.
mag	A numeric factor to scale the maximal thickness of paths.
...	additional arguments passed to the function <code>colorRamp</code> (in <code>vmd.cnapath</code> ).

### Details

This function generates a scaled sphere (communities) and stick (edges) representation of the community network along with the corresponding protein structure divided into chains, one chain for each community. The sphere radii are proportional to the number of community members and the edge widths correspond to network edge weights.

### Value

Two files are generated as output. A pdb file with the residue chains assigned according to the community and a text file containing The drawing commands for the community representation.

### Author(s)

Barry Grant

### References

Humphrey, W., Dalke, A. and Schulten, K., "VMD - Visual Molecular Dynamics" J. Molec. Graphics 1996, 14.1, 33-38.

### Examples

```
## Not run:

if (!requireNamespace("igraph", quietly = TRUE)) {
  message('Need igraph installed to run this example')
} else {

# Load the correlation network from MD data
attach(hivp)
```

```

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

# View cna
vmd.cna(net, pdb, launch=FALSE)
## within VMD set 'coloring method' to 'Chain' and 'Drawing method' to Tube

##-- From NMA
pdb.gdi = read.pdb("1KJY")
pdb.gdi = trim.pdb(pdb.gdi, inds=atom.select(pdb.gdi, chain="A", eley="CA"))
modes.gdi = nma(pdb.gdi)
cij.gdi = dccm(modes.gdi)
net.gdi = cna(cij.gdi, cutoff.cij=0.35)
#vmd.cna(net.gdi, pdb.gdi, alpha = 0.7, launch=TRUE)

detach(hivp)

}

## End(Not run)

```

---

vmd\_colors

*VMD Color Palette*


---

## Description

This function creates a character vector of the colors used by the VMD molecular graphics program.

## Usage

```
vmd_colors(n=33, picker=FALSE, ...)
```

## Arguments

n	The number of desired colors chosen in sequence from the VMD color palette (>=1)
picker	Logical, if TRUE a color wheel plot will be produced to aid with color choice.
...	Extra arguments passed to the rgb function, including alpha transparency.

## Details

The function uses the underlying 33 RGB color codes from VMD, See <http://www.ks.uiuc.edu/Research/vmd/>. Note that colors will be recycled if “n” > 33 with a warning issued. When ‘picker’ is set to “TRUE” a color wheel of the requested colors will be plotted to the currently active device.

## Value

Returns a character vector with color names.

**Author(s)**

Barry Grant

**References**Grant, B.J. et al. (2006) *Bioinformatics* **22**, 2695–2696.<http://www.ks.uiuc.edu/Research/vmd/>**See Also**[bwr.colors](#)**Examples**

```
## Generate a vector of 10 colors
clrs <- vmd_colors(10)
vmd_colors(4, picker=TRUE)
```

---

`wrap.tor`*Wrap Torsion Angle Data*

---

**Description**

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

**Usage**

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

**Arguments**

<code>data</code>	a numeric vector or matrix of torsion angle data as obtained from <code>torsion.xyz</code> .
<code>wrapav</code>	logical, if TRUE average structure is also ‘wrapped’
<code>avestruc</code>	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 ‘wrapping’ angular data such that the absolute difference of any of the observations from its column-wise mean is not greater than 180 degrees.

**Value**

A numeric vector or matrix of wrapped torsion angle data.

**Author(s)**

Karim ElSawy

**References**

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

**See Also**

[torsion.xyz](#)

---

write.crd

*Write CRD File*

---

**Description**

Write a CHARMM CARD (CRD) coordinate file.

**Usage**

```
write.crd(pdb = NULL, xyz = pdb$xyz, resno = NULL, resid = NULL,  
          eleno = NULL, elety = NULL, segid = NULL, resno2 = NULL, b = NULL,  
          verbose = FALSE, file = "R.crd")
```

**Arguments**

pdb	a structure object obtained from <a href="#">read.pdb</a> or <a href="#">read.crd</a> .
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:

[https://www.charmm.org/wiki//index.php/CHARMM:The\\_Basics](https://www.charmm.org/wiki//index.php/CHARMM:The_Basics).

**See Also**

[read.crd](#), [read.pdb](#), [atom.select](#), [write.pdb](#), [read.dcd](#), [read.fasta.pdb](#), [read.fasta](#)

**Examples**

```
## Not run:
# Read a PDB file
pdb <- read.pdb( "1bg2" )
summary(pdb)
# Convert to CHARMM format
new <- convert.pdb(pdb, type="charmm")
summary(new)
# Write a CRD file
write.crd(new, file="4charmm.crd")

## End(Not run)
```

---

write.fasta

*Write FASTA Formated Sequences*

---

**Description**

Write aligned or un-aligned sequences to a FASTA format file.

**Usage**

```
write.fasta(alignment=NULL, ids=NULL, seqs=alignment$ali, gap=TRUE, file, append = FALSE)
```

**Arguments**

alignment	an alignment list object with id and ali components, similar to that generated by <a href="#">read.fasta</a> .
ids	a vector of sequence names to serve as sequence identifiers
seqs	an sequence or alignment character matrix or vector with a row per sequence
gap	logical, if FALSE gaps will be removed.
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: <https://www.ncbi.nlm.nih.gov/BLAST/blastcgihelp.shtml>.

**Author(s)**

Barry Grant

**References**

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

**See Also**

[read.fasta](#), [read.fasta.pdb](#)

**Examples**

```
# PDB server connection required - testing excluded

## Read a PDB file
pdb <- read.pdb("1bg2")

## Extract sequence from PDB file
s <- aa321(pdb$seqres)          # SEQRES
a <- aa321(pdb$atom[pdb$calpha,"resid"]) # ATOM

## Write simple fasta file
#write.fasta( seqs=seqbind(s,a), file="eg.fa")
#write.fasta( ids=c("seqres","atom"), seqs=seqbind(s,a), file="eg.fa" )

outfile1 = file.path(tempdir(), "eg.fa")
write.fasta(list( id=c("seqres"),ali=s ), file = outfile1)
```

```
write.fasta(list( id=c("atom"),ali=a ), file = outfile1, append=TRUE)

## Align seqres and atom records
#seqaln(seqbind(s,a))

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

## Cut all but positions 130 to 245
aln$ali=aln$ali[,130:245]

outfile2 = file.path(tempdir(), "eg2.fa")
write.fasta(aln, file = outfile2)

invisible( cat("\nSee the output files:", outfile1, outfile2, sep="\n") )
```

---

write.mol2

*Write MOL2 Format Coordinate File*

---

## Description

Write a Sybyl MOL2 file

## Usage

```
write.mol2(mol, file = "R.mol2", append = FALSE)
```

## Arguments

mol	a MOL2 structure object obtained from <a href="#">read.mol2</a> .
file	the output file name.
append	logical, if TRUE output is appended to the bottom of an existing file (used primarily for writing multi-model files).

## Details

See examples for further details.

## Value

Called for its effect.

## Author(s)

Lars Skjaerven

## References

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

## Examples

```
# Read MOL2 file
mol <- read.mol2( system.file("examples/aspirin.mol2", package="bio3d") )

# Trim away H-atoms
mol <- trim(mol, "noh")

# Write new MOL2 file
#write.mol2(mol)
```

---

write.ncdf

*Write AMBER Binary netCDF files*

---

## Description

Write coordinate data to a binary netCDF trajectory file.

## Usage

```
write.ncdf(x, trjfile = "R.ncdf", cell = NULL)
```

## Arguments

x	A numeric matrix of xyz coordinates with a frame/structure per row and a Cartesian coordinate per column.
trjfile	name of the output trajectory file.
cell	A numeric matrix of cell information with a frame/structure per row and a cell length or angle per column. If NULL cell will not be written.

## Details

Writes an AMBER netCDF (Network Common Data Form) format trajectory file with the help of David W. Pierce's (UCSD) ncd4 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. <https://www.unidata.ucar.edu/software/netcdf/> <https://cirrus.ucsd.edu/~pierce/netcdf/> <https://ambermd.org/FileFormats.php#netcdf>

**See Also**

[read.dcd](#), [read.ncdf](#), [read.pdb](#), [write.pdb](#), [atom.select](#)

**Examples**

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

## Write to netCDF format
write.ncdf(trj, "newtrj.nc")

## Read trj
trj <- read.ncdf("newtrj.nc")

## End(Not run)
```

---

write.pdb

*Write PDB Format Coordinate File*

---

**Description**

Write a Protein Data Bank (PDB) file for a given 'xyz' Cartesian coordinate vector or matrix.

**Usage**

```
write.pdb(pdb = NULL, file = "R.pdb", xyz = pdb$xyz, type = NULL, resno = NULL,
  resid = NULL, eleno = NULL, elety = NULL, chain = NULL, insert = NULL,
  alt = NULL, o = NULL, b = NULL, segid = NULL, elesy = NULL, charge = NULL,
  append = FALSE, verbose = FALSE, chainter = FALSE, end = TRUE, sse = FALSE,
  print.segid = FALSE)
```

**Arguments**

pdb	a PDB structure object obtained from <a href="#">read.pdb</a> .
file	the output file name.
xyz	Cartesian coordinates as a vector or 3xN matrix.
type	vector of record types, i.e. "ATOM" or "HETATM", with length equal to length(xyz)/3.
resno	vector of residue numbers of length equal to length(xyz)/3.
resid	vector of residue types/ids of length equal to length(xyz)/3.
eleno	vector of element/atom numbers of length equal to length(xyz)/3.
elety	vector of element/atom types of length equal to length(xyz)/3.
chain	vector of chain identifiers with length equal to length(xyz)/3.
insert	vector of insertion code with length equal to length(xyz)/3.
alt	vector of alternate record with length equal to length(xyz)/3.
o	vector of occupancy values of length equal to length(xyz)/3.
b	vector of B-factors of length equal to length(xyz)/3.
segid	vector of segment id of length equal to length(xyz)/3.
elesy	vector of element symbol of length equal to length(xyz)/3.
charge	vector of atomic charge of length equal to length(xyz)/3.
append	logical, if TRUE output is appended to the bottom of an existing file (used primarily for writing multi-model files).
verbose	logical, if TRUE progress details are printed.
chainter	logical, if TRUE a TER line is inserted at termination of a chain.
end	logical, if TRUE END line is written.
sse	logical, if TRUE secondary structure annotations are written.
print.segid	logical, if FALSE segid will not be written.

**Details**

Only the xyz argument is strictly required. Other arguments assume a default poly-ALA C-alpha structure with a blank chain id, occupancy values of 1.00 and B-factors equal to 0.00.

If the input argument xyz is a matrix then each row is assumed to be a different structure/frame to be written to a "multimodel" PDB file, with frames separated by "END" records.

**Value**

Called for its effect.

**Note**

Check that: (1) chain is one character long e.g. "A", and (2) resno and eleno do not exceed "9999".

**Author(s)**

Barry Grant with contributions from Joao Martins.

**References**

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

For a description of PDB format (version3.3) see:

<http://www.wwpdb.org/documentation/format33/v3.3.html>.

**See Also**

[read.pdb](#), [read.dcd](#), [read.fasta.pdb](#), [read.fasta](#)

**Examples**

```
# PDB server connection required - testing excluded

# Read a PDB file
pdb <- read.pdb( "1bg2" )

# Renumber residues
nums <- as.numeric(pdb$atom["resno"])
nums <- nums - (nums[1] - 1)

# Write out renumbered PDB file
outfile = file.path(tempdir(), "eg.pdb")
write.pdb(pdb=pdb, resno = nums, file = outfile)

invisible( cat("\nSee the output file:", outfile, sep = "\n") )
```

---

write.pir

*Write PIR Formated Sequences*

---

**Description**

Write aligned or un-aligned sequences to a PIR format file.

**Usage**

```
write.pir(alignment=NULL, ids=NULL, seqs=alignment$ali,
  pdb.file = NULL, chain.first = NULL, resno.first = NULL,
  chain.last = NULL, resno.last = NULL, file, append = FALSE)
```

**Arguments**

alignment	an alignment list object with id and ali components, similar to that generated by <a href="#">read.fasta</a> .
ids	a vector of sequence names to serve as sequence identifiers
seqs	an sequence or alignment character matrix or vector with a row per sequence
pdb.file	a vector of pdb filenames; For sequence, provide "".
chain.first	a vector of chain id for the first residue.
resno.first	a vector of residue number for the first residue.
chain.last	a vector of chain id for the last residue.
resno.last	a vector of residue number for the last residue.
file	name of output file.
append	logical, if TRUE output will be appended to file; otherwise, it will overwrite the contents of file.

**Value**

Called for its effect.

**Note**

PIR is required format for input alignment file to use Modeller. For a description of PIR format see: <https://salilab.org/modeller/manual/node501.html>.

**Author(s)**

Xin-Qiu Yao

**References**

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

**See Also**

[read.fasta](#), [read.fasta.pdb](#), [write.fasta](#)

**Examples**

```
# Needs MUSCLE installed - testing excluded

if(check.utility("muscle")) {

## Generate an input file for structural modeling of
## transducin G-alpha subunit using the template 3SN6_A

## Read transducin alpha subunit sequence
seq <- get.seq("P04695", outfile = tempfile())
```

```

## Read structure template
path = tempdir()
pdb.file <- get.pdb("3sn6_A", path = path, split = TRUE)
pdb <- read.pdb(pdb.file)

## Build an alignment between template and target
aln <- seqaln(seqbind(pdbseq(pdb), seq), id = c("3sn6_A", seq$id), outfile = tempfile())

## Write PIR format alignment file
outfile = file.path(tempdir(), "eg.pir")
write.pir(aln, pdb.file = c(pdb.file, ""), file = outfile)

invisible( cat("\nSee the output file:", outfile, sep = "\n" ) )

}

```

---

write.pqr

---

*Write PQR Format Coordinate File*


---

### Description

Write a PQR file for a given 'xyz' Cartesian coordinate vector or matrix.

### Usage

```

write.pqr(pdb = NULL, xyz = pdb$xyz, resno = NULL, resid = NULL, eleno =
NULL, eley = NULL, chain = NULL, o = NULL, b = NULL,
append = FALSE, verbose = FALSE, chainter = FALSE, file = "R.pdb")

```

### Arguments

pdb	a PDB structure object obtained from <a href="#">read.pdb</a> or <a href="#">read.pqr</a> .
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.
eley	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	atomic partial charge values of length equal to length(xyz)/3.
b	atomic radii values of length equal to length(xyz)/3.
append	logical, if TRUE output is appended to the bottom of an existing file (used primarily 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

PQR file format is basically the same as PDB format except for the fields of o and b. In PDB, these two fields are filled with ‘Occupancy’ and ‘B-factor’ values, respectively, with each field 6-column long. In PQR, they are atomic ‘partial charge’ and ‘radii’ values, respectively, with each field 8-column long.

Only the xyz argument is strictly required. Other arguments assume a default poly-ALA C-alpha structure with a blank chain id, atomic charge values of 0.00 and atomic radii equal to 1.00.

If the input argument xyz is a matrix then each row is assumed to be a different structure/frame to be written to a “multimodel” PDB file, with frames separated by “END” records.

## Value

Called for its effect.

## Note

Check that: (1) chain is one character long e.g. “A”, and (2) resno and eleno do not exceed “9999”.

## Author(s)

Barry Grant with contributions from Joao Martins.

## References

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

For a description of PDB format (version3.3) see:

<http://www.wwpdb.org/documentation/format33/v3.3.html>.

## See Also

[read.pqr](#), [read.pdb](#), [write.pdb](#), [read.dcd](#), [read.fasta.pdb](#), [read.fasta](#)

## Examples

```
# PDB server connection required - testing excluded

# Read a PDB file
pdb <- read.pdb( "1bg2" )

# Write out in PQR format
outfile = file.path(tempdir(), "eg.pqr")
write.pqr(pdb=pdb, file = outfile)

invisible( cat("\nSee the output file:", outfile, sep = "\n" ) )
```

# Index

## \* IO

- [aln2html](#), 19
- [as.pdb](#), 23
- [get.seq](#), 124
- [read.all](#), 231
- [read.cif](#), 233
- [read.crd](#), 235
- [read.crd.amber](#), 236
- [read.crd.charmm](#), 237
- [read.dcd](#), 239
- [read.fasta](#), 241
- [read.fasta.pdb](#), 242
- [read.mol2](#), 244
- [read.ncdf](#), 247
- [read.pdb](#), 249
- [read.pdcBD](#), 252
- [read.pqr](#), 254
- [read.prmtop](#), 257
- [write.crd](#), 300
- [write.fasta](#), 301
- [write.mol2](#), 303
- [write.ncdf](#), 304
- [write.pdb](#), 305
- [write.pir](#), 307
- [write.pqr](#), 309

## \* analysis

- [cna](#), 54
- [community.aln](#), 64
- [community.tree](#), 66
- [dccm.enma](#), 81
- [dccm.gnm](#), 82
- [dccm.nma](#), 84
- [dccm.pca](#), 85
- [deformation.nma](#), 89
- [filter.cmap](#), 105
- [filter.dccm](#), 106
- [fluct.nma](#), 114
- [geostas](#), 118
- [inspect.connectivity](#), 135

- [nma](#), 153
- [nma.pdb](#), 154
- [nma.pdbs](#), 158

## \* classes

- [is.mol2](#), 137
- [is.pdb](#), 138
- [is.select](#), 139
- [is.xyz](#), 140

## \* datasets

- [aa.index](#), 7
- [aa.table](#), 9
- [atom.index](#), 27
- [elements](#), 100
- [example.data](#), 104
- [sdENM](#), 266

## \* documentation

- [bio3d-package](#), 6

## \* hplot

- [blast.pdb](#), 41
- [hclustplot](#), 127
- [plot.bio3d](#), 191
- [plot.cmap](#), 194
- [plot.cna](#), 196
- [plot.core](#), 198
- [plot.dccm](#), 200
- [plot.dmat](#), 203
- [plot.enma](#), 205
- [plot.fasta](#), 207
- [plot.fluct](#), 208
- [plot.geostas](#), 210
- [plot.hmmer](#), 211
- [plot.nma](#), 214
- [plot.pca](#), 216
- [plot.pca.loadings](#), 218
- [plot.rmsip](#), 219

## \* manip

- [orient.pdb](#), 162
- [rle2](#), 260

## \* multivariate

- pca.pdbs, 168
- pca.tor, 169
- pca.xyz, 171
- \* **runs**
  - rle2, 260
- \* **utilities**
  - aa123, 10
  - aa2index, 11
  - aa2mass, 12
  - angle.xyz, 21
  - as.fasta, 22
  - as.select, 26
  - atom.select, 28
  - atom2ele, 31
  - atom2mass, 33
  - atom2xyz, 34
  - basename.pdb, 35
  - bhattacharyya, 36
  - binding.site, 38
  - blast.pdb, 41
  - bounds, 44
  - bwr.colors, 46
  - cat.pdb, 47
  - chain.pdb, 48
  - check.utility, 50
  - cmap, 52
  - cnapath, 57
  - com, 60
  - combine.select, 62
  - consensus, 68
  - conserv, 69
  - convert.pdb, 71
  - core.cmap, 73
  - core.find, 74
  - cov.nma, 78
  - covoverlap, 79
  - dccm, 80
  - dccm.xyz, 87
  - diag.ind, 91
  - difference.vector, 92
  - dist.xyz, 93
  - dm, 94
  - dssp, 97
  - entropy, 101
  - filter.identity, 108
  - filter.rmsd, 110
  - fit.xyz, 111
  - formula2mass, 115
  - gap.inspect, 116
  - get.pdb, 122
  - get.seq, 124
  - hmmer, 129
  - inner.prod, 133
  - is.gap, 136
  - lbio3d, 142
  - load.enmff, 142
  - mask, 145
  - mktrj, 146
  - motif.find, 148
  - mustang, 149
  - normalize.vector, 161
  - overlap, 163
  - pairwise, 165
  - pca, 166
  - pca.array, 167
  - pca.pdbs, 168
  - pca.tor, 169
  - pca.xyz, 171
  - pdb.annotate, 173
  - pdb2aln, 175
  - pdb2aln.ind, 177
  - pdbaln, 180
  - pdffit, 182
  - pdbs2pdb, 183
  - pdbs2sse, 185
  - pdbsseq, 186
  - pdbsplit, 187
  - pfam, 189
  - print.cna, 220
  - print.core, 222
  - print.fasta, 223
  - print.xyz, 224
  - project.pca, 225
  - rgyr, 259
  - rmsd, 261
  - rmsf, 263
  - rmsip, 264
  - seq2aln, 267
  - seqaln, 269
  - seqaln.pair, 272
  - seqbind, 273
  - seqidentity, 275
  - setup.ncore, 276
  - sip, 277
  - sse.bridges, 278
  - store.atom, 279



- struct.aln, 280
- torsion.pdb, 282
- torsion.xyz, 284
- trim, 286
- trim.mol2, 288
- trim.pdbs, 289
- trim.xyz, 290
- unbound, 291
- uniprot, 292
- var.xyz, 294
- vec2resno, 295
- wrap.tor, 299
- \* **utility**
  - identify.cna, 132
  - layout.cna, 140
  - network.amendment, 151
  - prune.cna, 226
  - vmd, 296
  - vmd\_colors, 298
  - .print.fasta.ali (print.fasta), 223
- aa.index, 7, 9, 12, 13
- aa.table, 9
- aa123, 6, 10
- aa2index, 11
- aa2mass, 9, 12, 14, 155, 159, 171
- aa321, 6, 187
- aa321 (aa123), 10
- aanma, 14, 18, 144
- aanma.pdbs, 16, 17
- aln2html, 6, 19, 208
- amsm.xyz, 118
- amsm.xyz (geostas), 118
- angle.xyz, 21
- annotation (example.data), 104
- as.fasta, 22
- as.pdb, 23, 234, 237, 251, 258
- as.pdb.mol2, 246, 288
- as.select, 26, 30, 63
- as.xyz, 24, 291
- as.xyz (is.xyz), 140
- atom.index, 9, 13, 27, 27, 32, 34
- atom.select, 6, 15, 23, 25, 27, 28, 29, 32, 33, 35, 38, 39, 48, 49, 53, 62, 63, 72, 96, 119, 123, 125, 126, 139, 145, 146, 155, 163, 186–188, 234, 235, 237, 238, 240, 248, 251, 253, 256, 258, 286–288, 295, 301, 305
- atom.select.mol2, 246, 288
- atom2ele, 27, 31, 33, 34, 116
- atom2mass, 13, 32, 33, 61, 116
- atom2xyz, 34
- basename, 36
- basename.pdb, 35
- bhattacharyya, 19, 36, 80, 160, 265, 278
- binding.site, 38
- bio3d (bio3d-package), 6
- bio3d-package, 6
- biunit, 6, 40, 250, 251
- blast.pdb, 6, 41, 42, 125, 131, 212, 271, 293
- bounds, 44, 291, 292
- bounds.sse, 45, 179
- build.hessian, 6, 14, 144, 155
- build.hessian (nma.pdb), 154
- bwr.colors, 46, 299
- cat.pdb, 47, 234, 251
- chain.pdb, 48
- check.utility, 50
- clean.pdb, 50
- cm.colors, 47
- cmap, 52, 105, 107, 195
- cna, 6, 54, 58, 59, 64, 65, 67, 107, 152, 221, 227
- cna.dccm, 59
- cnapath, 6, 57
- col2rgb, 47
- colorRamp, 297
- colors, 47
- com, 60
- combine.select, 30, 62
- community.aln, 64
- community.tree, 66, 152
- consensus, 6, 68, 103, 109, 275
- conserv, 6, 69
- contour, 202, 204
- convert.pdb, 71
- cor, 88
- core (example.data), 104
- core.cmap, 73
- core.find, 6, 74, 74, 104, 181, 199, 222, 232, 243
- cov.enma, 18, 160
- cov.enma (cov.nma), 78
- cov.nma, 78
- covoverlap, 19, 37, 79, 80, 160, 265, 278
- cutree, 128

- dccm, *6, 53, 80, 86, 88, 107, 119, 120, 145, 146, 229*  
 dccm.egnm (dccm.gnm), *82*  
 dccm.enma, *18, 80, 81, 81, 83, 86, 88, 160*  
 dccm.gnm, *82*  
 dccm.nma, *6, 16, 80–83, 84, 86, 88, 107, 157*  
 dccm.pca, *80, 81, 85, 88*  
 dccm.xyz, *80, 81, 86, 87, 107*  
 deformation.nma, *6, 89*  
 diag, *91*  
 diag.ind, *91*  
 difference.vector, *92, 164*  
 dirname, *36*  
 dist, *53, 94*  
 dist.xyz, *53, 93*  
 dm, *39, 53, 94, 94, 135, 195, 203, 204*  
 dssp, *97, 98, 179, 186, 192–195, 201, 202, 213, 279, 283*  
 dssp.pdb, *98*  
  
 edge.betweenness.community, *56*  
 eigen, *157*  
 elements, *9, 27, 32, 34, 100*  
 entropy, *6, 70, 101, 109, 208, 275*  
 example.data, *104*  
  
 fastgreedy.community, *56*  
 ff.aaenm (load.enmff), *142*  
 ff.aaenm2 (load.enmff), *142*  
 ff.anm (load.enmff), *142*  
 ff.calpha (load.enmff), *142*  
 ff.pfanm (load.enmff), *142*  
 ff.reach (load.enmff), *142*  
 ff.sdenm (load.enmff), *142*  
 filled.contour, *202, 204*  
 filter.cmap, *105*  
 filter.dccm, *106*  
 filter.identity, *108, 275*  
 filter.rmsd, *110*  
 fit.xyz, *6, 53, 74–76, 87, 93, 111, 140, 163, 171, 180–183, 225, 226, 232, 243, 259, 262, 264, 275*  
 fluct.nma, *6, 16, 114, 157*  
 formula2mass, *32, 115*  
  
 gap.inspect, *6, 116, 135, 137, 290*  
 geostas, *6, 118, 119, 211*  
 get.blast (blast.pdb), *41*  
 get.pdb, *6, 43, 122, 125, 188*  
  
 get.seq, *6, 22, 23, 124, 131, 190, 271, 293*  
 get.shortest.paths, *58, 59*  
 gnm, *83, 125*  
 gnm.pdbs, *127*  
 graph.adjacency, *56*  
 gray, *47*  
  
 hclust, *6, 110, 118, 119, 128, 207*  
 hclustplot, *127*  
 hivp (example.data), *104*  
 hmmer, *6, 43, 129, 190, 212*  
 hsv, *47*  
  
 identify, *133*  
 identify.cna, *132*  
 igraph.plotting, *133, 141, 197, 221*  
 image, *202, 204, 220*  
 infomap.community, *56*  
 inner.prod, *133, 161*  
 inspect.connectivity, *135*  
 is.gap, *6, 136*  
 is.mol2, *137*  
 is.pdb, *138*  
 is.pdbs (is.pdb), *138*  
 is.select, *139*  
 is.xyz, *140, 225*  
  
 kinesin (example.data), *104*  
 kmeans, *118, 119*  
  
 layout.cna, *140*  
 lbio3d, *142*  
 load.enmff, *15, 16, 142, 156, 157*  
 lower.tri, *91*  
  
 mask, *145*  
 matrix, *91*  
 mktrj, *118–120, 146*  
 mktrj.enma, *18, 160*  
 mktrj.nma, *6, 16, 157*  
 mktrj.pca, *6, 172*  
 mono.colors (bwr.colors), *46*  
 motif.find, *148*  
 mustang, *6, 149*  
  
 network.amendment, *67, 151*  
 nma, *6, 18, 78, 82, 84, 89, 90, 114, 115, 119, 120, 134, 144, 148, 153, 153, 161, 164, 169, 206, 215, 220, 265*  
 nma.pdb, *16, 147, 153, 154, 155, 156, 160, 172*

- nma.pdbs, *6, 18, 119, 147, 148, 153, 158, 159, 205, 208, 209, 294*  
normalize.vector, *134, 161*
- orient.pdb, *6, 162*  
overlap, *6, 92, 157, 163, 220, 265*
- pairwise, *6, 165*  
palette, *47*  
pca, *147, 148, 153, 166, 169, 172, 265*  
pca.array, *167, 213, 214*  
pca.pdbs, *6, 167, 168, 172*  
pca.tor, *6, 167, 169, 172, 226, 285*  
pca.xyz, *6, 86, 147, 164, 166–170, 171, 172, 216–218, 226*  
pdb.annotate, *104, 173*  
pdb.pfam (pdb.annotate), *173*  
pdb2aln, *175, 177, 178*  
pdb2aln.ind, *176, 177*  
pdb2sse, *45, 46, 179*  
pdbaln, *6, 23, 28, 53, 73, 75, 95, 98, 104, 119, 137, 138, 147, 151, 153, 158–160, 167, 169, 177, 180, 182–184, 186, 223, 224, 228, 271, 281, 290*  
pdbfit, *6, 104, 182*  
pdbs (example.data), *104*  
pdbs2pdb, *183*  
pdbs2sse, *185*  
pdbseq, *6, 10, 149, 181, 186, 250, 257*  
pdbsplit, *123, 187*  
pfam, *131, 189*  
plot.bio3d, *6, 99, 191, 195, 202, 209, 215, 217*  
plot.blast, *6, 43, 131, 212*  
plot.blast (blast.pdb), *41*  
plot.cmap, *105, 194*  
plot.cna, *6, 56, 65, 133, 141, 196, 227*  
plot.cnapath (cnapath), *57*  
plot.communities, *133, 141, 197*  
plot.core, *6, 76, 198, 222*  
plot.dccm, *6, 80–84, 86, 107, 200, 211, 213, 214*  
plot.default, *193, 195, 202*  
plot.dendrogram, *128*  
plot.dmat, *96, 195, 202, 203*  
plot.ecna (plot.cna), *196*  
plot.ecnapath (cnapath), *57*  
plot.enma, *18, 160, 205*  
plot.fasta, *151, 207, 271*  
plot.fluct, *206, 208*  
plot.geostas, *120, 210*  
plot.hclust, *128*  
plot.hmmer, *211*  
plot.igraph, *133, 141, 197*  
plot.matrix.loadings, *213*  
plot.nma, *6, 214*  
plot.pca, *6, 170, 172, 216, 218*  
plot.pca.loadings, *6, 170, 218*  
plot.rmsip, *219*  
plotb3, *206*  
plotb3 (plot.bio3d), *191*  
points, *215*  
polygon, *209*  
print, *155, 159*  
print.cna, *220*  
print.cnapath (cnapath), *57*  
print.core, *199, 222*  
print.default, *260*  
print.enma (nma.pdbs), *158*  
print.fasta, *223*  
print.geostas (geostas), *118*  
print.igraph, *221*  
print.mol2 (read.mol2), *244*  
print.nma (nma.pdb), *154*  
print.pca (pca.xyz), *171*  
print.pdb (read.pdb), *249*  
print.prmtop (read.prmtop), *257*  
print.rle2 (rle2), *260*  
print.select (atom.select), *28*  
print.sse (dssp), *97*  
print.xyz, *224*  
project.pca, *172, 225*  
prune.cna, *226*  
pymol, *228*  
pymol.dccm, *6, 80, 81*  
pymol.modes, *6, 148*  
pymol.pdbs, *181, 243*
- rbind, *273, 274*  
read.all, *17–19, 53, 95, 181, 231, 243*  
read.cif, *233*  
read.crd, *25, 30, 98, 99, 235, 258, 300, 301*  
read.crd.amber, *23, 235, 236*  
read.crd.charmm, *235, 237, 237*  
read.dcd, *6, 22, 30, 72, 75, 98, 99, 111, 113, 118, 120, 140, 225, 234, 235, 238, 239, 248, 250, 251, 253, 255, 256, 264, 285, 301, 305, 307, 310*

- read.fasta, 6, 10, 12, 20, 68–70, 72, 102,  
 103, 109, 116, 117, 123–125, 137,  
 149, 151, 175, 177, 181, 187, 190,  
 208, 223, 224, 231, 232, 238, 241,  
 242, 243, 251, 253, 256, 267–269,  
 271–275, 290, 301, 302, 307, 308,  
 310  
 read.fasta.pdb, 6, 28, 53, 70, 72–76, 95, 98,  
 110, 111, 113, 116, 117, 119, 123,  
 125, 137, 138, 147, 151, 153,  
 158–160, 175, 177, 181–184, 186,  
 223, 224, 228, 238, 241, 242, 251,  
 253, 256, 259, 262, 264, 268, 271,  
 273, 280, 290, 301, 302, 307, 308,  
 310  
 read.mol2, 23, 29, 138, 244, 288, 303  
 read.ncdf, 6, 25, 30, 75, 98, 99, 118, 120,  
 140, 225, 234, 235, 237, 247, 250,  
 251, 258, 290, 305  
 read.pdb, 6, 10, 14, 22, 24, 25, 27, 28, 30, 32,  
 34, 35, 39, 40, 45, 47–49, 51, 53, 58,  
 61, 63, 71, 75, 95–97, 99, 111–113,  
 119, 120, 123, 125, 138, 140, 145,  
 147, 153, 155, 162, 163, 179, 181,  
 183, 184, 186–188, 192, 194, 201,  
 213, 225, 231, 232, 234, 235, 237,  
 238, 240, 243, 246, 248, 249, 249,  
 250, 256, 258, 259, 262, 274, 279,  
 283, 285–287, 290, 291, 295, 300,  
 301, 305–307, 309, 310  
 read.pdb2 (read.pdb), 249  
 read.pdcBD, 252  
 read.pqr, 254, 309, 310  
 read.prmtop, 23, 29, 30, 99, 234, 235, 237,  
 251, 257, 257  
 regexpr, 149  
 rgb, 47  
 rgyr, 259  
 rle2, 260  
 rmsd, 6, 110, 111, 113, 183, 259, 261, 281  
 rmsf, 6, 208, 209, 263  
 rmsip, 6, 19, 37, 157, 160, 164, 220, 264, 278  
 rot.lsqr, 6, 163, 262, 281  
 rot.lsqr (fit.xyz), 111  
 rtb (aanma), 14  
  
 sdENM, 266  
 seq2aln, 176, 178, 267  
 seqaln, 6, 20, 23, 43, 109, 131, 137, 151,  
 175–177, 181, 207, 208, 223, 224,  
 267, 268, 269, 272–274, 281  
 seqaln.pair, 176, 178, 272  
 seqbind, 22, 23, 268, 269, 271–273, 273  
 seqidentity, 6, 109, 166, 275  
 setup.ncore, 276  
 sip, 19, 37, 80, 160, 265, 277  
 sse.bridges, 278  
 store.atom, 279  
 str.igraph, 221  
 stride, 98, 179, 192–195, 201, 202, 213, 283  
 stride (dssp), 97  
 struct.aln, 280  
 summary.cna, 56, 67, 152, 227  
 summary.cna (print.cna), 220  
 summary.cnpath (cnpath), 57  
 summary.pdb, 6  
 summary.pdb (read.pdb), 249  
  
 t.test, 209  
 torsion.pdb, 6, 22, 99, 282, 285  
 torsion.xyz, 6, 22, 99, 170, 283, 284, 300  
 transducin (example.data), 104  
 trim, 286  
 trim.mol2, 246, 288  
 trim.pdb, 30, 32, 33, 48, 49, 63, 234, 251, 290  
 trim.pdbs, 287, 289  
 trim.xyz, 287, 290  
  
 unbound, 291  
 uniprot, 131, 190, 292  
 upper.tri, 91  
  
 var.pdbs (var.xyz), 294  
 var.xyz, 159, 294  
 vec2resno, 295  
 vmd, 296  
 vmd.cna, 56, 59, 65, 227  
 vmd.cnpath, 59  
 vmd\_colors, 47, 298  
  
 walktrap.community, 56  
 wrap.tor, 285, 299  
 write.crd, 235, 238, 300  
 write.fasta, 6, 20, 274, 301, 308  
 write.mol2, 246, 288, 303  
 write.ncdf, 6, 248, 304

write.pdb, [6](#), [30](#), [48](#), [49](#), [72](#), [123](#), [147](#), [163](#),  
[188](#), [234](#), [235](#), [238](#), [240](#), [248](#), [251](#),  
[253](#), [256](#), [295](#), [301](#), [305](#), [305](#), [310](#)

write.pir, [307](#)

write.pqr, [256](#), [309](#)

xyz2atom (atom2xyz), [34](#)

xyz2z.pca (project.pca), [225](#)

z2xyz.pca (project.pca), [225](#)