Package ‘BioPhysConnectoR’

February 19, 2015

Version 1.6-10
Date 2013-01-09
Title BioPhysConnectoR
Author Franziska Hoffgaard <hoffgaard@bio.tu-darmstadt.de>, with
  contributions from Philipp Weil <weil@bio.tu-darmstadt.de> and
  Kay Hamacher <hamacher@bio.tu-darmstadt.de>
Maintainer Frank Keul <keul@bio.tu-darmstadt.de>
Depends R (>= 2.11.0), snow, matrixcalc
Description Utilities and functions to investigate the relation
  between biomolecular structures, their interactions, and the
  evolutionary information revealed in sequence alignments of
  these molecules.
License GPL-2
Repository CRAN
Date/Publication 2013-01-09 17:22:13
NeedsCompilation yes

R topics documented:

  BioPhysConnectoR-package ........................................... 2
  aa2num .............................................................. 4
  build.contacts .......................................................... 5
  build.hess ............................................................ 6
  build.interact .......................................................... 7
  extractPDB ............................................................. 8
  fnorm ................................................................. 9
  get.bfacs ............................................................. 10
  get.contact.list ...................................................... 11
  get.entropy .......................................................... 12
  get.freqs ............................................................ 13
  get.mie .............................................................. 15
  get.svd .............................................................. 16
BioPhysConnectoR-package

Index

BioPhysConnectoR-package

Description

Functions to investigate the relation between biomolecular structures, their interactions, and the evolutionary information contained within sequence alignments of such molecules.

Details

Package: BioPhysConnectoR
Version: 1.6-7
Date: 2011-11-03
Depends: R (>= 2.11.0), snow, matrixcalc
License: GPL 2
Packaged: 2010-23-09 13:50:31 UTC; weil
Built: R 2.12.1; x86_64-pc-linux-gnu; 2011-11-03 16:55:01 UTC; unix

Index:

- aa2num
- BioPhysConnectoR-package
- build.contacts
- build.hess
- build.interact
- build.invhess
- extractPDB
- fnorm
- get.bfacs

Conversion of Amino Acids into Integer Values
BioPhysConnectoR
Determine the Contact Map and Distance Matrices
Construct the Hessian Matrix
Compute the Interaction Matrix
Compute the Covariance Matrix / Inverse Hessian Matrix
Extract Data from a PDB-File
Frobenius Norm of Two Matrices
Determine B factors
get.contact.list Returns a List of Contacts for a given Contact Map
get.cov Compute the Covariance Matrix / Inverse Hessian Matrix
get.entropy Compute the Sequence Entropy for an Alignment
get.freqs Compute the Frequencies in an Alignment
get.mie Mutual Information
get.svd Singular Value Decomposition
lbpc List the Functions of the BioPhysConnectoR Package
mat.norm Normalisation of a Matrix
mat.read Read Matrix Data from a File
mat.sort Sort a Matrix According to a Specified Column
mat.write Writes Matrix Data to a File
read.fasta Reads aligned or un-aligned sequences from a FASTA format file
scppc Self-Consistent Pair Contact Probability Approximation
show.code Output of the Amino Acid Coding Scheme
sim Compute the Correlation Matrix and B Factors for a List of PDBs
sinc Computed Elastic Network Models for Switched-Off-List of Contacts
sims Apply a List of Different Amino Acid Sequences

Author(s)
Franziska Hoffgaard <hoffgaard@bio.tu-darmstadt.de>, with contributions from Philipp Weil <weil@bio.tu-darmstadt.de> and Kay Hamacher <hamacher@bio.tu-darmstadt.de>

Maintainer: Franziska Hoffgaard <hoffgaard@bio.tu-darmstadt.de>

References
aa2num  
*Conversion of Amino Acids into Integer Values*

Description

A sequence containing standard amino acids is converted into a sequence of integer values. An offset for the code can be specified.

Usage

```r
aa2num(seq, offset = 0, code = 0:19, verbose = FALSE)
```

Arguments

- `seq` : string vector containing a sequence of standard amino acids
- `offset` : offset, added to the specified code
- `code` : integer vector of the values to be assigned to the amino acids
- `verbose` : logical, if TRUE the specific encoding is printed

Details

The default values for the code are the integer values in the range 0 to 19. It is possible to use another numerical code for the 20 amino acids. The default coding order of the amino acids is: CYS, MET, PHE, ILE, LEU, VAL, TRP, TYR, ALA, GLY, THR, SER, ASN, GLN, ASP, GLU, HIS, ARG, LYS, PRO. Each amino acid is converted to a single element according to the code-vector. The offset `offset` is added. Both sequences with three-letter and one-letter code can be handled.

Value

Returns a sequence of integer values according to the given numerical code.

Note

The function assigns values only for the standard amino acids. This function includes source code of the bio3d package.

Author(s)

Franziska Hoffgaard

References


See Also

`show.code`
**build.contacts**

**Examples**

```r
seq <- c("MET", "GLY", "PRO", "LYS", "ASN")
aa2num(seq)
```

---

**Description**

Computation of a binary matrix specifying the contacts between each two amino acids $i$ and $j$ in respect to their spatial distance defined by their coordinates. Distance matrices for $\Delta x$, $\Delta y$ and $\Delta z$ are computed as well as the matrix containing the squared distances for all amino acid pairs.

**Usage**

```r
build.contacts(n, cuts, xyz)
```

**Arguments**

- **n**: length of the amino acid sequence
- **cuts**: squared cutoff
- **xyz**: matrix with the x-, y- and z-coordinates of all $C_\alpha$ atoms of the protein

**Details**

If the squared distance between two $C_\alpha$ atoms of amino acids $i$ and $j$ is smaller than or equal to `cuts`, we assume a contact. In the contact map the value at indices $[i, j]$ and $[j, i]$ is set to 1. Otherwise, the two $C_\alpha$ atoms are not in contact, and the value is set to 0. Per definition an atom is not in contact with itself. The contact map is a symmetric matrix. The matrices in `$deltas` ($dx$, $dy$ and $dz$ and $ds$) are symmetric matrices as well. The number of contacts between distinct amino acids is stored in `$cnr`.

**Value**

Return value is a list with the following components:

- **$cm**: contact map
- **$deltas**: list with distance matrices for x-, y- and z-direction as well as for the squared distance between all pairs of $C_\alpha$ atoms
- **$cnr**: number of contacts

**Author(s)**

Franziska Hoffgaard
**build.hess**

**Construct the Hessian Matrix**

Description

The Hessian matrix is computed using the interaction matrix, the contact map, and the coordinate differences.

Usage

```r
build.hess(cm, im, deltas)
```

Arguments

- `cm` contact map
- `im` interaction map
- `deltas` list of difference matrices computed with `build.contacts`

Details

For a sequence of length \( n \) a \( 3n \times 3n \) matrix is computed. Only if amino acids are in contact (\( cm \) entry equals 1) values for the Hessian matrix are computed as proposed in Hamacher (2006). If amino acids are not in contact their respective matrix entries equal 0. Values for amino acids in contact depend on coordinate differences and interaction strengths.

Value

Return value is the Hessian Matrix.

Author(s)

Franziska Hoffgaard

References


See Also

`build.interact`, `build.contacts`
build.interact

Compute the Interaction Matrix

Description
Calculation of the interaction strength between all amino acids of a protein.

Usage
build.interact(cseq, mj1, mj2 = mj1, d, alpha = 82)

Arguments
- `cseq` (coded) amino acid sequence
- `mj1` matrix for intrachain interactions between amino acids
- `mj2` matrix for interchain interactions between amino acids
- `d` vector of chain lengths
- `alpha` strength of the peptide bond

Details
Per default only one matrix with interaction strengths between amino acids is used to compute the interaction map. But it is possible to differentiate between intrachain [Miyazawa and Jernigan (1996)] and interchain [Keskin et al. (1998)] interactions by using an additional interaction matrix. Both matrices are included in the package and can be used directly. The value of `d` specifies the lengths of the different chains in the sequence.

Per definition there is no interaction between an amino acid and itself, therefore the diagonal is set to zero. All entries that represent peptide bonds between two amino acids are set to `alpha` (only entries in the off-diagonals). The input of `cseq` can be a numeric sequences [0, 19] as well as a string vector of standard amino acid sequences in one or three-letter code.

Value
Return value is a symmetric matrix specifying the interactions between the amino acids. If `n` is the length of the sequence, the matrix dimension results in `n × n`.

Note
Choosing one’s own matrices for inter- and intrachain interactions requires some thought: this function only deals with the standard amino acids thus it requires always symmetric, 20 × 20 matrices.

Author(s)
Franziska Hoffgaard
extractPDB

This function reads a PDB file and extracts data from it.

Usage

extractPDB(file.name, verbose = TRUE)

Arguments

file.name name of the PDB file
verbose logical, if TRUE prints the output from both functions read.pdb() and atom.select().

Details

The input PDB file is read. All $C_\alpha$ atoms are extracted. For each $C_\alpha$ atom the x-, y- and z-coordinates as well as the amino acid type and the B factor are extracted. The sequence length is compared to the number of $C_\alpha$ atoms in the PDB. For each chain of the protein the length is computed.
Value

Returns a list with the following components:

- **pdb**: list of class “pdb” as originally extracted by `read.pdb()` in the **bio3d** package
- **seq**: sequence according to the “SEQRES” entries of the PDB file
- **lseq**: length of the extracted sequence seq
- **lca**: number of $\text{C}_\alpha$ atoms
- **caseq**: sequence data for the $\text{C}_\alpha$ selection
- **coords**: matrix of coordinates for each $\text{C}_\alpha$ atom, the rows are the $\text{C}_\alpha$’s, the columns are $x, y, z$
- **b**: B factor for each $\text{C}_\alpha$
- **chains**: integer vector with the lengths of the different chains in the protein sequence

Note

Alternate and insert records in the PDB file are ignored. When removing alternate records a message is printed. Please check the PDB file.

If the number of $\text{C}_\alpha$ atoms `lca` and the length of the extracted sequence `seq` differ, a message is printed. It is not advisable to use any other parts of the BioPhysConnectoR-package until you have found an appropriate solution to deal with this problem in the pdb-file.

This function includes source code of the **bio3d** package.

Author(s)

Franziska Hoffgaard

References


Examples

```r
pdb<-extractPDB(system.file("1KZK.pdb", package = "BioPhysConnectoR"))
```

---

<table>
<thead>
<tr>
<th>fnorm</th>
<th>Frobenius Norm of Two Matrices</th>
</tr>
</thead>
</table>

Description

Computation of the Frobenius norm of two matrices as the sum of the squared differences between these matrices.
get.bfacs

Usage
fnorm(mat1, mat2)

Arguments
mat1 numerical matrix
mat2 numerical matrix

Value
Return value is the Frobenius norm.

Note
Both input matrices must have the same dimensions.

Author(s)
Franziska Hoffgaard

Examples
a<-matrix(runif(15, 1, 7), 5, 3)
b<-matrix(rnorm(15, 1, 7), 5, 3)
fn<-fnorm(a, b)

get.bfacs Determine B factors

Description
The function calculates B factors from the inverse Hessian matrix.

Usage
get.bfacs(covmat)

Arguments
covmat inverse Hesse matrix

Details
The B factors for each Cα atom are computed from the diagonal of the covariance matrix by summing up the corresponding entries.
get.contact.list

Returns a List of Contacts for a given Contact Map

Description

From a given contact map a list of contacts is computed.

Usage

get.contact.list(cm, d = NULL, single = TRUE, val = 1)

Arguments

cm binary matrix which specifies which amino acid position are in contact
d vector specifying the last amino acids of each chain
single bool, if TRUE only indices i < j are returned
val numeric value, indices of matrix entries with this value are returned

Value

Return value is a vector with the B factors for each \( C_\alpha \).

Author(s)

Franziska Hoffgaard

See Also

build.invhess

Examples

#extract the example pdb
pdb<-extractPDB(system.file("1KZK.pdb", package = "BioPhysConnectoR"))

#build the contact matrix for a given squared cutoff of 169
bc<-build.contacts(pdb$1ca, 169, pdb$coords)

#for this example we use the fictitious value 12 as interaction strength
interaction.mat<-build.interact(cseq = rep(0, 198), mj1 = matrix(12, 20, 20),
d = pdb$chains)

#compute the inverse hessian matrix
cov.mat<-get.cov(cm = bc$cm, im = interaction.mat, deltas = bc$deltas)
bfac<-get.bfac(cov.mat)
Details

All amino acid pairs with \( \text{cm}[i,j] \equiv \text{val} \) are extracted from the cm. Each pair is listed once \([i,j] = [j,i]\) (if \(\text{single} = \text{TRUE}\)) with \(i < j\). If \(d\) is specified the contacts of the off-diagonal (peptide bonds) are left out, otherwise they will be returned as well. The values of \(d\) are the indices of the off-diagonal entries that do not represent a peptide bond.

Value

Return value is a list with the indices of all pairs of amino acids that fulfill the above stated condition.

Author(s)

Franziska Hoffgaard

See Also

build.contacts.simc

Examples

```r
mat <- matrix(c(0, 0, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0), ncol = 5)
gc <- get.contact.list(mat)
```

---

**get.entropy**

*Compute the Sequence Entropy for an Alignment*

Description

Computes the sequence entropy of an alignment. It is possible to specify which characters to omit in the computation. The joint entropy is computed using get.entropy2p().

Usage

```r
gc <- get.entropy(aln, bool = FALSE, gapchar = "NOGAPCHAR",
                 verbose = FALSE)
gc <- get.entropy2p(aln, bool = FALSE, gapchar = "NOGAPCHAR",
                  verbose = FALSE)
```

Arguments

- **aln**
  - alignment matrix
- **bool**
  - logical, if TRUE gaps are ignored when computing the entropy of each column of the alignment
- **gapchar**
  - character vector containing the unique set of characters representing gaps in the amino acid sequence
- **verbose**
  - logical, TRUE for getting output messages
**get.freqs**

**Description**

One and two point frequencies of a specified column or specified columns are computed for a given alignment.

**Details**

The Shannon (1948) entropy for an alignment is computed as follows:

\[ H(X) = - \sum_{x \in X} p(x) \cdot \log_2(p(x)) \]

The joint entropy is computed for every possible column pair:

\[ H(X, Y) = - \sum_{x \in X} \sum_{y \in Y} p(x, y) \cdot \log_2(p(x, y)) \]

where \( X \) and \( Y \) are two columns of the alignment.

**Value**

Return value for `get.entropy()` is a vector containing the entropy for each column. Return value for `get.entropy2p()` is a matrix containing the joint entropies in the lower triangle.

**Author(s)**

Franziska Hoffgaard

**References**


**See Also**

`get.mie`

**Examples**

```r
h1<-get.entropy(aln, bool = TRUE, gapchar = "-")
h2<-get.entropy(aln)

h3<-get.entropy2p(aln)
```

**get.freqs**

*Compute the Frequencies in an Alignment*

**Description**

One and two point frequencies of a specified column or specified columns are computed for a given alignment.
Usage

freq1p(aln, i = NULL)

freq2p(i, aln, j2 = NULL, lett = NULL, cluster = NULL)

Arguments

aln    alignment
i      reference column for the frequency computation
j2     columns to compute the two point frequency together with column i
lett    character vector containing the unique alphabet of the alignment
cluster snow cluster object created with makeCluster()

Details

The columns j2 are the specified columns (not i) to compute the frequencies and the rows represent all possible two letter pairs according to the alphabet. The computation of freq2p() is parallelized using parLapply() from the package snow. If cluster is left at its default value the computation is carried out in serial.

Value

A matrix is returned for the freq2p(). For freq1p(), if no i is specified, a matrix containing the frequencies of the symbols in the alignment is returned. For a certain i a vector with the respective frequencies of the symbols is the result.

Author(s)

Franziska Hoffgaard, Philipp Weil

References


See Also

get.entropy, get.mie

Examples

seqa<-unlist(strsplit("PQITLWQRPLVTIKIGQQL",split=""))
seqb<-unlist(strsplit("PQITLWKRPLVTIRIGQQL",split=""))
seqc<-unlist(strsplit("PQITLWQRPLVTIKGQQL",split=""))
aln<-(matrix(c(seqa,seqb,seqc),nrow=3,byrow=TRUE)
f1<-(freq1p(aln)
f1a<-(freq1p(aln, 1)
f2<-(freq2p(1, aln, 2:10)
## Not run:
## Description

The joint information content (mutual information) for an alignment is computed. Considering the gap problem there are four ways to compute it.

### Usage

```r
get.mie(aln, method = "ORMI", gapchar = NULL, nullmod = NULL, logMI = FALSE)
```

### Arguments

- `aln`: matrix of which the mutual information will be computed
- `method`: method, that is used for the computation (see details)
- `gapchar`: symbols of the input matrix that should be handled as gaps. These symbols are omitted in the computations (see details)
- `nullmod`: integer specifying how many shuffle runs should be performed
- `logMI`: boolean, if TRUE the log(MI) will be calculated, default is FALSE

### Details

Methods: The mutual information (MI) is computed as follows:

\[
I(X, Y) = \sum_{x \in X} \sum_{y \in Y} p(x, y) \cdot \log_2 \left( \frac{p(x, y)}{p(x) \cdot p(y)} \right)
\]

\[
I(X, Y) = H(X) + H(Y) - H(X, Y)
\]

**ORMI** In the ORiginal MI gaps are treated simply as any other character.

**SUMI** The SUbset MI omits for each pair of columns all the rows with at least one gap character for the computation.

**DEMI** In the Delta Entropy MI the entropies for the columns are computed separately by leaving out any gap characters. The joint entropy also considers only rows without any gap character. DEMI follows as \(H(X) + H(Y) - H(X, Y)\).

**ESMI** The Enhanced Sampling MI omits the gap characters and rows with gaps for the computation of the probabilities. The information content is computed via the probabilities.

The null model is computed by shuffling each column content and computing the resulting MI. Over all shuffle runs the MI values in each entry of the matrix is averaged. The averaged squared MI values and the variance is computed as well.
get.svd

Value

Return value is the MI matrix per default. If a null model should be computed the returned value is a list of matrices.

\$mi  MI matrix
\$nullmodel  MI matrix of the null model
\$nullsquare  matrix of the averaged MI squared values for the null model
\$nullvar  matrix of the variance of each MI value over the whole shuffle run for the null model

Author(s)

Franziska Hoffgaard

See Also

get.entropy, get.entropy2p, freq1p, freq2p

Examples

seqa<-unlist(strsplit("PQITLWQRPLVTIKIGGL",split=""))
seqb<-unlist(strsplit("PQITLWKRPLVTIRIGGQ",split=""))
seqc<-unlist(strsplit("PQITLWQRPLVTIKIGGL",split=""))
a<-matrix(c(seqa,seqb,seqc),nrow=3,byrow=TRUE)
mi<-get.mie(a)
mi2<-get.mie(a, method = "SUMI", gapchar = ")")
mi_null<-get.mie(a,nullmod=100)
mi2_null<-get.mie(a, method = "SUMI", gapchar = ",nullmod=100)

get.svd  

Singular Value Decomposition

Description

Computation of the singular value decomposition for a matrix. Sorts the eigenvalues in ascending order by maintaining their original order in an index array.

Usage

get.svd(hessian.mat, linpack = TRUE)

Arguments

hessian.mat  input matrix is a Hessian matrix
linpack  logical, specifies whether LINPACK or LAPACK routines are used
invhess

Value

Return value is a list with the following components:

- `$v$` matrix with eigenvectors in each column
- `$indx$` index array for the sorted eigenvalues
- `$ev$` vector with the sorted eigenvalues
- `$u$` matrix with eigenvectors in each column

Note

This function uses the `svd()` function of R.

Author(s)

Franziska Hoffgaard

See Also

`mat.sort`, `build.hess`

Examples

```r
a <- matrix(round(runif(9, 1, 5)), 3, 3)
out <- get.svd(a)
```

invhess  Compute the Covariance Matrix / Inverse Hessian Matrix

Description

Computes the inverse Hessian matrix. The covariance matrix is computed as a pseudo-inverse derived from the eigenvalues and eigenvectors by a singular value decomposition (`get.svd()`) of the Hessian matrix. Otherwise, if neither the Hessian matrix nor the eigenvalues need to be stored, the inverse Hessian can directly be computed from the contact, interaction and distance matrices.

Usage

```r
build.invhess(svd_obj, singularity = 6)
get.cov(cm, im, deltas)
```
### Arguments

- **svd_obj**: svd object computed by `get.svd()` containing the eigenvector matrices, the eigenvalues and the index vector
- **singularity**: number of eigenvalues equal/close to zero due to symmetries
- **cm**: contact map for a protein
- **im**: matrix of interaction strengths between the amino acids of the protein
- **deltas**: difference matrices (x, y, z, squared) for all pairs of $C_{\alpha}$ atoms as derived from `build.contacts()`

### Details

The calculation of the matrix omits by default the first six eigenvalues, because of translational and rotational symmetry in the model. The computation depends on the eigenvalues and -vectors. The number of eigenvalues to omit in the calculation can be specified by `singularity`. If the number of eigenvalues equalling zero is unknown and should be determined, the parameter `singularity` can be set to `NULL`. The threshold for zero is set to $10^{-8}$.

### Value

Return value is the covariance matrix (also called inverse Hessian matrix).

### Author(s)

Franziska Hoffgaard

### References


### See Also

- `build.hess`
- `get.svd`

---

### Description

A shortcut for `ls("package:BioPhysConnectoR")` to get an overview of the implemented functions.

### Usage

```r
lbpc()
```

### Value

No return value. Prints the function names.
**mat.norm**

**Description**

A matrix is normalized by dividing each entry $[i, j]$ by the square root of the product of the diagonal entries $[i, i]$ and $[j, j]$. The input matrix should be a square matrix with positive diagonal entries.

**Usage**

```r
mat.norm(mat)
```

**Arguments**

- `mat`: numerical matrix to be normalized

**Value**

The normalized matrix is returned.

**Author(s)**

Franziska Hoffgaard

---

**mat.read**

**Description**

A matrix is constructed from a specified input file.

**Usage**

```r
mat.read(file.name, ij = FALSE, sym = FALSE)
```
Arguments

file.name  file name
ij  logical, if TRUE the format in the file is: \( i \ j \ value \), otherwise each row of the file represents one row in the matrix
sym  logical, if TRUE the matrix is symmetric, only important if \( ij = \) TRUE

Details

There are two ways a matrix can be specified in the input file. If each line represents a row in the matrix (default), the matrix can simply be read. Otherwise it is possible to have an input file with the form: \( i \ j \ value \), this means the value at position \([i, j]\) of the matrix is set to \( value \). If \( ij = \) TRUE and \( sym = \) TRUE only one half of the matrix needs to be provided.

Value

Return value is the matrix.

Author(s)

Franziska Hoffgaard

mat.sort  Sort a Matrix According to a Specified Column

Description

Sorts a matrix in respect to one or more specified columns by preserving its row context. If more than one columns is given the function uses the columns in the specified order for sorting.

Usage

`mat.sort(mat, sort, decreasing = FALSE)`

Arguments

mat  matrix to be sorted
sort  column indices
decreasing  logical, if TRUE the column is ordered decreasingly

Value

Returns the sorted matrix.

Author(s)

Franziska Hoffgaard
Examples

```r
mat <- matrix(data = as.integer(runif(16, 1, 20)), nrow = 4)

# Sort the matrix in respect to the second column of mat
m <- mat[order(mat[, 2]),]
```

```r
# Sorting mat according to more than one column
mat <- matrix(data = c(rep(3, 3), as.integer(runif(13, 1, 20))), nrow = 4)
m1 <- mat[order(mat[, c(1, 2)]),]
m2 <- mat[order(mat[, c(2, 1)]),]
```

---

**mat.write**

Writes Matrix Data to a File

**Description**

Matrix data are written to a specified output file.

**Usage**

```r
mat.write(mat, file.name, ij = FALSE, sym = FALSE, sparse=FALSE, formatted=TRUE)
```

**Arguments**

- `mat`: matrix which should be written
- `file.name`: file name
- `ij`: logical, if TRUE the format in the file will be: \(i\ j\ value\), otherwise each row of the file represents one row in the matrix
- `sym`: logical, if TRUE the matrix is symmetric, only important if \(ij = \text{TRUE}\)
- `sparse`: logical, if TRUE the matrix is written in sparse format (e.g. only non-zero values are written to the file), only important if \(ij = \text{TRUE}\)
- `formatted`: logical, if TRUE a blank line is inserted after all entries of one row, only important if \(ij = \text{TRUE}\)

**Details**

In general there are two ways the output file can be written. First the matrix can be written to the file as `write.table` does and second the file can be written with the form: \(i\ j\ value\), this means the value at position \([i, j]\). If the latter by \(ij = \text{TRUE}\) is selected, three more options can be chosen. If \(sym = \text{TRUE}\) the output file just contains the upper triangle and the diagonal of the matrix. If \(sparse = \text{TRUE}\) only non-zero values are written. If \(formatted = \text{TRUE}\) a blank line separates the entries of different rows.

**Value**

No values are returned.
**read.fasta**

**Author(s)**
Philipp Weil

**See Also**
mat.read

---

**read.fasta**

*Read FASTA formatted Sequences*

**Description**
Read aligned or un-aligned sequences from a FASTA format file.

**Usage**

```r
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.

**Note**
For a description of FASTA format see: [http://www.ebi.ac.uk/help/formats_frame.html](http://www.ebi.ac.uk/help/formats_frame.html). When reading alignment files, the dash `\` is interpreted as the gap character.

**Author(s)**
Barry Grant

**References**
Description

The model of the Self-Consistent Pair Contact Probability (SCPCP) (Micheletti et al., 2001; Hamacher et al., 2006) computes equilibrium properties of structures with known native states. For a given contact map, extracted from a PDB file or artificially created, the fraction of native contacts, the free and internal energies are computed as well as the degree to which an amino acid is in its native state conformation. The maximum number of iteration and the preferred accuracy for the approximation can be specified.

Usage

\[
\text{scpcp}(T, R, \text{cm}, \text{pstart} = 0.5, \text{maxiter} = 2000, \text{chains=}\text{NULL}, \text{maxtol} = 1e-11, \\
\text{file} = \text{NULL}, \text{im} = \text{NULL})
\]

Arguments

\begin{itemize}
  \item \text{T} \quad \text{temperature}
  \item \text{R} \quad \text{distance cutoff between current and native state}
  \item \text{cm} \quad \text{contact map}
  \item \text{pstart} \quad \text{initial probability value}
  \item \text{chains} \quad \text{vector denoting the chain lengths}
  \item \text{maxiter} \quad \text{maximum number of iterations}
  \item \text{maxtol} \quad \text{tolerance}
  \item \text{file} \quad \text{output file name}
  \item \text{im} \quad \text{interaction matrix}
\end{itemize}

Details

Results of each iteration as well as the final results are written into the given output file.

Value

Returns a list with the following components

\begin{itemize}
  \item \$\text{free} \quad \text{free energy}
  \item \$\text{intern} \quad \text{internal energy}
  \item \$\text{entropy} \quad \text{entropy}
  \item \$\text{q} \quad \text{fraction of native contacts}
  \item \$\text{bfacs} \quad \text{vector containing the B factors}
  \item \$\text{pi} \quad \text{probability vector}
  \item \$\text{gmat} \quad \text{resulting G matrix}
  \item \$\text{iter} \quad \text{number of iterations}
  \item \$\text{err} \quad \text{deviation of probabilities}
\end{itemize}
Author(s)
Franziska Hoffgaard

References

Examples
```r
## Not run:
p<-extractPDB(system.file("1KZK.pdb", package = "BioPhysConnectorR"))
cm<-build.contacts(n = p$lca, xyz = p$coords, cuts = 169)$cm
chains<-p$chains
im<-build.interact(cseq=p$caseq,d=chains,mj=matrix(0.05,20,20),alpha=1)
res<-scpccp(cm = cm, maxtol = 10^-10, T = 5, R = 0.8, im = im,
maxiter = 10, pstart = 0.5, chains=chains)
## End(Not run)
```

---

**Output of the Amino Acid Coding Scheme**

Description
Displays an encoding table for the standard amino acids.

Usage
```r
show.code(code = 0:19, offset = 0)
```

Arguments
- code : vector of 20 integer values used for coding the standard amino acids
- offset : offset added to the values of code

Details
This function shows the coding order of the standard amino acid as used in `aa2num()`. The offset is included.

Value
No values are returned.
**Description**

For a given list of PDB-files the respective covariance matrices and resulting B factors are computed.

**Usage**

```r
sim(pdzs, mj1 = NULL, mj2 = NULL, mj.avg = FALSE, alpha = 82, 
cuts = 169, path = getwd(), cluster = NULL)
```

**Arguments**

- `pdzs`: list of PDB file names
- `mj1`: matrix for the intrachain interaction strengths
- `mj2`: matrix for the interchain interaction strengths
- `mj.avg`: logical, if TRUE only the average value of the interaction matrix is used as value for the interaction of any two amino acids
- `alpha`: strength of the peptide bond
- `cuts`: squared distance cutoff
- `path`: path to the output files
- `cluster`: snow cluster object created with `makeCluster()`

**Details**

Each PDB file is extracted. All features necessary for the computation of the covariance matrix in the elastic network model are computed. Both the covariance matrix and the B factors are computed and written to a file. The computation is parallelized using parLapply() from the package `snow`. If `cluster` is left at its default value the computation is carried out in serial.

**Value**

No values are returned. The B factors and inverse Hessian matrices are written into files.
Description

For each entry in the contact list the contact will be broken and the resulting covariance matrix and new B factors will be computed in the elastic network model. Furthermore the Frobenius norms between the original and the new covariance matrix can be evaluated.

Usage

```r
simc(pdb, mj1 = NULL, mj2 = NULL, mj.avg = FALSE, cl = NULL,
     alpha = 82, cuts = 169, path = getwd(), inv2file = FALSE,
     bfacs = TRUE, frob = TRUE, loc = NULL, norm = FALSE,
     file = NULL, cluster = NULL)
```
Arguments

- **pdb**: file name of the PDB
- **mj1**: matrix for the intrachain interaction strengths
- **mj2**: matrix for the interchain interaction strengths
- **mj.avg**: logical, if TRUE only the average value of the matrices is used as value for the interaction of any two amino acids
- **cl**: optional contact list to process
- **alpha**: strength of the peptide bond
- **cuts**: squared cutoff distance
- **path**: path to the output files
- **inv2file**: logical, if TRUE the inverse Hessian matrix is written to a file, otherwise it will not be stored
- **bfacs**: logical, if TRUE, the B factors are written to a file, otherwise they will not be stored
- **frob**: logical, if TRUE, the Frobenius norm is computed
- **loc**: dimensions i1, j1, i2, j2 for a matrix subset of which the frobenius norm should be computed
- **norm**: logical, if TRUE the Frobenius norm is computed for the normalized matrices
- **file**: personalized file name prefix
- **cluster**: snow cluster object created with `makeCluster()`

Details

If no contact list is given, the full contact list is extracted from the PDB-file. Each contact (except covalent contacts) in the list is broken and the corresponding covariance matrix and B factors are computed. Those can be written into files. A user-defined contact list can be specified as well. For the computation of the Frobenius norm, different regions can be specified in `loc` as matrix. Each row determines a region [i1,j1,i2,j2] to be used for the norm. The routine is parallelized for the list of contacts using `parLapply()` from the package snow. If `cluster` is left at its default value the computation is carried out in serial.

Value

No values are returned.

Author(s)

Franziska Hoffgaard, Philipp Weil

References

Apply a List of Different Amino Acid Sequences

Description
For a set of sequences given in an alignment and a corresponding PDB file the covariance matrix for each sequence is computed based on the given molecular structure. The latter can be directed into a file. The Frobenius norms are computed upon request.

Usage
sims(pdb, alignment, mj1 = NULL, mj2 = NULL, mj.avg = FALSE, alpha = 82, cuts = 169, path = getwd(), mimethod = "ORMI", gapchar = "NOGAPCHAR", inv2file = FALSE, bfacs = TRUE, frob = TRUE, loc = NULL, norm = FALSE, cluster = NULL)

Arguments
- pdb: PDB file
- alignment: alignment file in fasta format
- mj1: matrix for the intrachain interaction strengths
- mj2: matrix for the interchain interaction strengths
- mj.avg: logical, if TRUE only the average value of each matrix is used as value for the interaction of any two amino acids
- alpha: strength of the peptide bond
- cuts: squared distance cutoff
- path: path to the output files
- mimethod: method for the computation of the mutual information

Examples
## Not run:
c <- matrix(c(3,1,4,1,5,1,9,1,10,1,11,1,24,1,66,1,67,1,68,1),ncol=2,byrow=TRUE)
out <- simc(system.file("1KZK.pdb", package = "BioPhysConnectoR"), cuts = 169, cl=c1)

## Cluster example
makeCluster(2)->clu
out<-simc(system.file("1KZK.pdb", package = "BioPhysConnectoR"), cuts = 169, cl=cl, cluster=clu)
stopCluster(clu)

## End(Not run)
sims

- **gapchar**: character vector denoting gaps in the alignment
- **inv2file**: logical, if TRUE, the inverse Hessian matrix is written to a file
- **bfacs**: logical, if TRUE, the B factors are written to a file
- **frob**: logical, if TRUE, the Frobenius norm is computed
- **loc**: dimensions i1, j1, i2, j2 for a matrix subset of which the Frobenius norm should be computed
- **norm**: logical, if TRUE the Frobenius norm is computed for the normalized matrices
- **cluster**: snow cluster object created with `makeCluster()`

**Details**
For an alignment the sequence entropy and mutual information is computed. Furthermore for each sequence in the alignment and the structure information from the PDB file the covariance matrix and B factors can be computed and the output can be written to files. The computation is parallelized using `parLapply()` from the package `snow`. If `cluster` is left at its default value the computation is carried out in serial.

**Value**
Return value is a list with the following components:

- `$entropy`: sequence entropy of the alignment
- `$mi`: mutual information of the alignment

If the Frobenius norm is computed, the value(s) will be returned as well:

- `$res`: vector consisting of: row number, sequence name, Frobenius norm

**Note**
Make sure your alignment contains the IUPAC standard amino acids only without any gap characters.
This function includes source code of the `bio3d` package.

**Author(s)**
Franziska Hoffgaard, Philipp Weil

**References**

**See Also**
get.entropy, get.mie, sim, simc
Examples

## Not run:
sims(system.file("1KZK.pdb",package="BioPhysConnectoR"), system.file("align.fasta", package="BioPhysConnectoR"))

## Cluster example
clu<-makeCluster(2)
sims(system.file("1KZK.pdb",package="BioPhysConnectoR"), system.file("align.fasta", package="BioPhysConnectoR",cluster = clu))
stopCluster(clu)

## End(Not run)
Index

*Topic IO
  extractPDB, 8
  mat.read, 19
  mat.write, 21
  read.fasta, 22

*Topic package
  BiophysConnector-package, 2

*Topic utilities
  aa2num, 4
  build.contacts, 5
  build.hess, 6
  build.interact, 7
  fnorm, 9
  get.bfacs, 10
  get.contact.list, 11
  get.entropy, 12
  get.freqs, 13
  get.mie, 15
  get.svd, 16
  invhess, 17
  lbpc, 18
  mat.norm, 19
  mat.sort, 20
  scpcp, 23
  show.code, 24
  sim, 25
  simc, 26
  sims, 28

  aa2num, 4, 8, 25

BiophysConnectorR
  (BiophysConnectorR-package), 2

BiophysConnectorR-package, 2
  build.contacts, 5, 6, 12
  build.hess, 6, 17, 18
  build.interact, 6, 7
  build.invhess, 11
  build.invhess (invhess), 17

extractPDB, 8

fnorm, 9
freq1p, 16
freq1p (get.freqs), 13
freq2p, 16
freq2p (get.freqs), 13

get.bfacs, 10
get.contact.list, 6, 11
get.cov (invhess), 17
get.entropy, 12, 14, 16, 29
get.entropy2p, 16
get.entropy2p (get.entropy), 12
get.freqs, 13
get.mie, 13, 14, 15, 29
get.svd, 16, 18

invhess, 17

lbpc, 18

mat.norm, 19
mat.read, 19, 22
mat.sort, 17, 20
mat.write, 21

read.fasta, 22

scpcp, 23
show.code, 4, 24
sim, 25, 28, 29
simc, 12, 26, 29
sims, 26, 28, 28