Package 'MDaRes'

August 25, 2026

Title MD Analysis of Residue Properties Using Structural Alphabets

Version 0.0.2

Description Analyze molecular dynamics (MD) simulation data using structural alphabets. Protein local conformations from molecular simulations are encoded in a compressed string representation. Residue-level UniProt annotations can be directly retrieved. Comparative analysis tools are available to investigate conformational variability, coordinated motions, and differences between systems (e.g. wild-type vs mutant; bound vs unbound). Methodological details for the structural alphabet analysis can be found in Pandini et al. (2013) <doi:10.1093/bioinformatics/btt326>.

License GPL-3

Encoding UTF-8

RoxygenNote 7.3.3

Depends R (>= 4.1.0)

Imports bio3d, DescTools, SOMMD, tools, dplyr, tidyr, stringr, UniprotR, Rcpp

LinkingTo Rcpp, RcppArmadillo

Suggests httr, future, future.apply, curl, testthat (>= 3.0.0)

Config/testthat/edition 3

NeedsCompilation yes

Author Nancy D'Arminio [aut] (ORCID: <https://orcid.org/0000-0002-8065-2142>), Anna Marabotti [aut] (ORCID: <https://orcid.org/0000-0002-9510-5205>), Alessandro Pandini [aut, cph, cre] (ORCID: <https://orcid.org/0000-0002-4158-233X>)

Maintainer Alessandro Pandini <alessandro.pandini@gmail.com>

Repository CRAN

Date/Publication 2026-08-25 15:10:09 UTC

Contents

`add_active_site` *Retrieve and Format Active Site Annotations from UniProt*

Description

Fetches active site annotations (`ft_act_site`) from UniProt for a given protein accession code and structures them into a dataframe.

Usage

```
add_active_site(accession_code)
```

Arguments

accession_code	Character. The UniProt accession code of the protein (e.g., "P12345").

Details

This function queries UniProt for active site annotations using the GetProteinAnnontate function from the UniprotR package. It extracts the positions of active sites and their functional notes, then structures them into a dataframe.

If no annotations are found, the function returns an empty dataframe.

Value

A dataframe with the following columns:

- **Position**: Numeric. The residue position of the active site.
- **ActiveSite**: Character. A description of the active site function.

Examples

```
if (interactive()) {
active_sites_df <- add_active_site("P12345")
head(active_sites_df)
}
```

---

add_binding_site *Retrieve and Format Ligand Binding Site Annotations from UniProt*

Description

Fetches ligand binding site annotations (ft_binding) from UniProt for a given protein accession code and structures them into a dataframe.

Usage

```
add_binding_site(accession_code)
```

Arguments

accession_code	Character. The UniProt accession code of the protein (e.g., "P12345").

Details

This function queries UniProt for ligand binding site annotations using the `GetProteinAnnontate` function from the `UniprotR` package. It extracts ligand names and their corresponding residue positions, then structures them into a dataframe.

If no annotations are found, the function returns an empty dataframe.

Value

A dataframe with the following columns:

- **Position**: Numeric. The position(s) of the ligand binding site annotation.
- **LigandBindingSite**: Character. The name of the ligand binding site.

Examples

```
if (interactive()) {
binding_sites_df <- add_binding_site("P12345")
head(binding_sites_df)
}
```

`add_disulfide_bond` *Retrieve and Format Disulfide Bond Annotations from UniProt*

Description

Fetches disulfide bond annotations (`ft_disulfid`) from UniProt for a given protein accession code and structures them into a dataframe.

Usage

```
add_disulfide_bond(accession_code)
```

Arguments

`accession_code`	Character. The UniProt accession code of the protein (e.g., "P12345").

Details

This function queries UniProt for disulfide bond annotations using the `GetProteinAnnontate` function from the `UniprotR` package. It parses entries labeled with `"DISULFID"`, extracts residue positions provided either as single positions (e.g., `"150"`) or ranges (e.g., `"99..231"`), and expands ranges into per-residue rows.

Character strings equal to `"NA"` are converted to proper `NA` values. If no annotations are found (i.e., the retrieved field is `NULL`, `NA`, or empty), the function returns an empty dataframe.

Value

A dataframe with the following columns:

- **Position**: Numeric. The residue position(s) involved in the disulfide bond.
- **DisulfideBond**: Character. A label for the detected bond (set to "Bridge").

Examples

```
if (interactive()) {
disulfide_df <- add_disulfide_bond("P12345")
head(disulfide_df)
}
```

---

add_domain *Retrieve and Format Domain Annotations from UniProt*

Description

Fetches domain annotations (ft_domain) from UniProt for a given protein accession code and structures them into a dataframe.

Usage

```
add_domain(accession_code)
```

Arguments

accession_code	Character. The UniProt accession code of the protein (e.g., "P12345").

Details

This function queries UniProt for domain annotations using the GetProteinAnnontate function from the UniprotR package. It extracts domain names and their corresponding residue positions, then structures them into a dataframe.

If no annotations are found, the function returns an empty dataframe.

Value

A dataframe with the following columns:

- **Position**: Numeric. The position(s) of the domain annotation.
- **Domain**: Character. The name of the protein domain.

Examples

```
if (interactive()) {
domain_df <- add_domain("P12345")
head(domain_df)
}
```

`add_glycosilation` *Retrieve and Format Glycosylation Annotations from UniProt*

Description

Fetches glycosylation site annotations (`ft_carbohyd`) from UniProt for a given protein accession code and structures them into a dataframe.

Usage

```
add_glycosilation(accession_code)
```

Arguments

`accession_code` Character. The UniProt accession code of the protein (e.g., `"P12345"`).

Details

This function queries UniProt for glycosylation site annotations using the `GetProteinAnnontate` function from the `UniprotR` package. It extracts the positions of glycosylation sites and their functional notes, then structures them into a dataframe.

If no annotations are found, the function returns an empty dataframe.

Value

A dataframe with the following columns:

- **Position**: Numeric. The residue position of the glycosylation site.
- **Glycosylation**: Character. A description of the glycosylation modification.

Examples

```
if (interactive()) {
glycosylation_df <- add_glycosilation("P12345")
head(glycosylation_df)
}
```

`add_lipidation` *Retrieve and Format Lipidation Annotations from UniProt*

Description

Fetches lipidation site annotations (`ft_lipid`) from UniProt for a given protein accession code and structures them into a dataframe.

Usage

```
add_lipidation(accession_code)
```

Arguments

`accession_code` Character. The UniProt accession code of the protein (e.g., `"P12345"`).

Details

This function queries UniProt for lipidation site annotations using the `GetProteinAnnontate` function from the `UniprotR` package. It extracts the positions of lipidation sites and their functional notes, then structures them into a dataframe.

If no annotations are found, the function returns an empty dataframe.

Value

A dataframe with the following columns:

- **Position**: Numeric. The residue position of the lipidation site.
- **Lipidation**: Character. A description of the lipidation modification.

Examples

```
if (interactive()) {
lipidation_df <- add_lipidation("P12345")
head(lipidation_df)
}
```

`add_location` *Retrieve and Format Location Annotations from UniProt*

Description

Fetches subcellular location annotations (`ft_intramem`, `ft_transmem`, `ft_topo_dom`) from UniProt for a given protein accession code and structures them into a dataframe.

Usage

```
add_location(accession_code)
```

Arguments

`accession_code` Character. The UniProt accession code of the protein (e.g., `"P12345"`).

Details

This function queries UniProt for subcellular location annotations using the `GetProteinAnnontate` function from the `UniprotR` package. It extracts the location type (e.g., `"INTRAMEM"`, `"TRANSMEM"`, `"TOPO_DOM"`) and the corresponding residue positions, then structures them into a dataframe.

If no annotations are found, the function returns an empty dataframe.

Value

A dataframe with the following columns:

- **Position**: Numeric. The position(s) of the location annotation.
- **Location**: Character. The type of location with an associated note (e.g., `"TRANSMEM: Helical"`).

Examples

```
if (interactive()) {
location_df <- add_location("P12345")
head(location_df)
}
```

`add_modified_residue` *Retrieve and Format Modified Residue Annotations from UniProt*

Description

Fetches modified residue annotations (`ft_mod_res`) from UniProt for a given protein accession code and structures them into a dataframe.

Usage

```
add_modified_residue(accession_code)
```

Arguments

`accession_code` Character. The UniProt accession code of the protein (e.g., `"P12345"`).

Details

This function queries UniProt for modified residue site annotations using the `GetProteinAnnontate` function from the `UniprotR` package. It extracts the positions of modified residues and their functional notes, then structures them into a dataframe.

If no annotations are found, the function returns an empty dataframe.

Value

A dataframe with the following columns:

- **Position**: Numeric. The residue position of the modification.
- **ModifiedResidue**: Character. A description of the residue modification.

Examples

```
if (interactive()) {
modified_residue_df <- add_modified_residue("P12345")
head(modified_residue_df)
}
```

`add_molecule_processing` *Retrieve and Format Molecule Processing Annotations from UniProt*

Description

Fetches molecule processing annotations (`ft_chain`) from UniProt for a given protein accession code and structures them into a dataframe.

Usage

```
add_molecule_processing(accession_code)
```

Arguments

`accession_code` Character. The UniProt accession code of the protein (e.g., `"P12345"`).

Details

This function queries UniProt for molecule processing annotations using the `GetProteinAnnontate` function from the `UniprotR` package. It extracts the molecule processing type and the corresponding residue positions, then structures them into a dataframe.

If no annotations are found, the function returns an empty dataframe.

Value

A dataframe with the following columns:

- **Position**: Numeric. The position(s) of the molecule processing event.
- **MoleculeProcessing**: Character. The type of molecule processing (e.g., `"CHAIN"`).

Examples

```
if (interactive()) {
molecule_processing_df <- add_molecule_processing("P12345")
head(molecule_processing_df)
}
```

add_motive *Retrieve and Format Motif Annotations from UniProt*

Description

Fetches motif annotations (ft_motif) from UniProt for a given protein accession code and structures them into a dataframe.

Usage

```
add_motive(accession_code)
```

Arguments

accession_code Character. The UniProt accession code of the protein (e.g., "P12345").

Details

This function queries UniProt for motif annotations using the GetProteinAnnontate function from the UniprotR package. It extracts the motif type and the corresponding residue positions, then structures them into a dataframe.

If no annotations are found, the function returns an empty dataframe.

Value

A dataframe with the following columns:

- **Position**: Numeric. The position(s) of the motif annotation.
- **Motive**: Character. The type of motif.

Examples

```
if (interactive()) {
motif_df <- add_motive("P12345")
head(motif_df)
}
```

`add_secondary_structure` *Retrieve and Format Secondary Structure Annotations from UniProt*

Description

Fetches secondary structure annotations (`ft_turn`, `ft_helix`, and `ft_strand`) from UniProt for a given protein accession code and structures them into a dataframe.

Usage

```
add_secondary_structure(accession_code)
```

Arguments

`accession_code` Character. The UniProt accession code of the protein (e.g., `"P12345"`).

Details

This function queries UniProt for secondary structure annotations (turns, helices, and strands) using the `GetProteinAnnontate` function from the `UniprotR` package. It extracts the structure type and corresponding residue positions, then structures them into a dataframe.

If no annotations are found, the function returns an empty dataframe.

Value

A dataframe with the following columns:

- **Position**: Numeric. The residue position of the secondary structure element.
- **SecondaryStructure**: Character. The type of secondary structure (TURN, HELIX, STRAND).

Examples

```
if (interactive()) {
secondary_structure_df <- add_secondary_structure("P12345")
head(secondary_structure_df)
}
```

`add_uniprot_annotation` *Retrieve and Merge UniProt Annotations*

Description

Fetches various annotations from UniProt (e.g., variants, domains, secondary structure) and merges them into a single dataframe for a given protein accession code.

Usage

```
add_uniprot_annotation(accession_code, total_ranking = NULL)
```

Arguments

`accession_code`	Character. The UniProt accession code of the protein (e.g., "P12345").
`total_ranking`	Optional dataframe. If provided, the function will return annotation information aggregated over the fragments defined by the start-end positions in the ranking.

Details

This function calls multiple UniProt annotation functions (`add_variants`, `add_domain`, `add_location`, etc.) and merges their results into a unified dataframe, ensuring missing values are replaced with meaningful defaults.

If a UniProt API request fails or an annotation is unavailable, the respective column is filled with default values.

If `total_ranking` is provided, it must contain the columns `start`, `end`, `entropy`, `median_MI`, and `collectivity`. The function will expand each fragment into its constituent residue positions, merge them with UniProt annotations, and return a dataframe where annotations are concatenated for each fragment.

Value

A dataframe.

If `total_ranking` is not provided, each row represents a residue position with the following columns:

- **Position**: Numeric. Residue position.
- **anPosition**: Character. Combined annotation position (e.g., "P12345_1").
- **Residue**: Character. Residue identity (e.g., "A", "G", "L").
- **Variant**: Character. Mutation information or "no_variant" if absent.
- **Location**: Character. Membrane location or "EXTRAMEM" if absent.
- **Motive**: Character. Motif annotation or "no_motive" if absent.
- **Domain**: Character. Structural domain or "no_domain" if absent.
- **LigandBindingSite**: Character. Ligand binding site or "no_ligand" if absent.

- **ActiveSite**: Character. Active site information or "no_active_site" if absent.
- **Glycosylation**: Character. Glycosylation details or "no_glycans" if absent.
- **Lipidation**: Character. Lipidation details or "no_lipid" if absent.
- **ModifiedResidue**: Character. Post-translational modification details or "no_modifications" if absent.
- **DisulfideBond**: Character. Disulfide bond information or "no_bridge" if absent.
- **SecondaryStructure**: Character. Secondary structure type or "unknown" if absent.

If `total_ranking` is provided, each row corresponds to a fragment and includes:

- **start**, **end**, **entropy**, **median_MI**, **collectivity**
- plus concatenated annotation columns for all positions in the fragment (e.g., Residues, Variants, etc.)

Examples

```
if (interactive()) {
ranking_file <- system.file(
  "extdata", "example_total_ranking.RDS", package = "MDaRes"
)
ranking <- readRDS(ranking_file)

annotated_fragments <- add_uniprot_annotation(
  accession_code = "P69905",
  total_ranking = ranking
)
head(annotated_fragments)
}
```

`add_variants` *Retrieve and Format Protein Variants from UniProt*

Description

Fetches protein variant annotations from UniProt for a given accession code and structures them into a dataframe.

Usage

```
add_variants(accession_code)
```

Arguments

`accession_code` Character. The UniProt accession code of the protein (e.g., `"P12345"`).

Details

This function queries UniProt for variant annotations (`ft_variant`) using the `GetProteinAnnontate` function from the `UniprotR` package. It extracts the variant position, mutation information, and UniProt variant ID, then structures them into a dataframe.

The function ensures proper handling of missing values and returns an empty dataframe if no variants are found.

Value

A dataframe with the following columns:

- **Position**: Numeric. The position of the variant in the protein sequence.
- **Variant**: Character. The UniProt variant ID and the mutation change (e.g., `"VAR_123456 : A -> T"`).

Examples

```
if (interactive()) {
variant_df <- add_variants("P12345")
head(variant_df)
}
```

---

colnames_substring *Extract Substring from Column Names*

Description

This function removes a specific prefix from a string and returns the modified string. It is typically used to clean column names by removing unnecessary prefixes.

Usage

```
colnames_substring(string)
```

Arguments

`string`	A character vector representing column names or strings containing the prefix "Frag: ".

Value

A character vector with the prefix "Frag: " removed from each element.

create_annotation_df *Retrieve UniProt Protein Sequence and Format as Dataframe*

Description

Retrieves the amino acid sequence from UniProt for a given protein accession code and formats it into a structured dataframe.

Usage

```
create_annotation_df(accession_code)
```

Arguments

`accession_code` Character. The UniProt accession code of the protein (e.g., `"P12345"`).

Details

This function queries UniProt to fetch the full protein sequence using the `GetSequences` function from the `UniprotR` package. The sequence is then split into individual residues and structured into a dataframe where each row represents a residue with its sequential position.

Value

A dataframe with the following columns:

- **Position**: Numeric. The residue position in the sequence.
- **anPosition**: Character. The annotated position formatted as `"ACCESSION_POSITION"`.
- **Residue**: Character. The single-letter amino acid code at the given position.

do_freq_diff_plot *Save frequency-difference plot to PNG*

Description

Wrapper around `freq_diff_plot` that saves the plot as a PNG file.

Usage

```
do_freq_diff_plot(
  first_freq,
  second_freq,
  output_name,
  width = 1800,
  height = 800,
  res = 120,
  ...
)
```

Arguments

first_freq	A shannon_list object (from shannon_entropy).
second_freq	A shannon_list object (from shannon_entropy).
output_name	Character. Output file path **without** extension (".png" is appended).
width, height	Integer. PNG size in pixels.
res	Integer. PNG resolution (ppi).
...	Additional arguments passed to freq_diff_plot (e.g., zlim, n_ticks).

Value

Invisibly returns the frequency-difference matrix.

---

do_freq_plot *Generate Frequency Profile Plot and Save to PNG*

Description

Generates a structural sequence profile plot and saves it as a PNG file.

Usage

```
do_freq_plot(shannon_output, output_name)
```

Arguments

shannon_output	A list of class shannon_list, as returned by shannon_entropy().
output_name	Character. The filename (without extension) for the output image.

Details

The function calls freq_plot() to generate the plot and saves it as a PNG file.

Value

The function does not return anything but creates a PNG file.

Examples

```
shannon_file <- system.file(
  "extdata", "example_shannon_entropy.RDS", package = "MDaRes"
)
shannon_output <- readRDS(shannon_file)

output_base <- tempfile("frequency-plot-")
output_file <- paste0(output_base, ".png")
```

```
do_freq_plot(shannon_output, output_base)
file.exists(output_file)
unlink(output_file)
```

`do_MI_plot` *Generate Mutual Information Matrix Plot and Save to PNG*

Description

Generates a heatmap of the mutual information matrix and saves it as a PNG file.

Usage

```
do_MI_plot(MI_matrix, output_name)
```

Arguments

`MI_matrix`	A list of class `MI_list`, as returned by `MImat()`.
`output_name`	Character. The filename (without extension) for the output image.

Details

The function calls `MI_plot()` to generate the plot and saves it as a PNG file.

Value

The function does not return anything but creates a PNG file.

Examples

```
mi_file <- system.file(
  "extdata", "example_MI.RDS", package = "MDaRes"
)
MI_output <- readRDS(mi_file)

output_base <- tempfile("MI-plot-")
output_file <- paste0(output_base, ".png")

do_MI_plot(MI_output, output_base)
file.exists(output_file)
unlink(output_file)
```

do_shannon_entropy_diff_plot *Save Shannon Entropy Difference Plot to PNG*

Description

Wrapper around shannon_entropy_diff_plot that opens a PNG device, draws the plot, and closes the device. This is useful for batch workflows where plots must be written to disk without manual interaction.

Usage

```
do_shannon_entropy_diff_plot(
  first_shannon,
  second_shannon,
  output_name,
  overlay = TRUE,
  width = 2600,
  height = 1200,
  res = 220
)
```

Arguments

first_shannon	A shannon_list object containing the first Shannon entropy output (expects $entropy and, optionally, $residue_numbering).
second_shannon	A shannon_list object containing the second Shannon entropy output to be compared against first_shannon.
output_name	Character. Output path **without** file extension. The function will write paste0(output_name, ".png").
overlay	Logical. If TRUE (default), overlay the two original entropies on the right axis, as implemented in shannon_entropy_diff_plot.
width	Integer. PNG width in pixels. Defaults to 1800.
height	Integer. PNG height in pixels. Defaults to 800.
res	Integer. PNG resolution in dots per inch. Defaults to 200. Increase for publication-quality output (e.g., 300).

Details

The function uses png to create the file device and ensures that dev.off is called even if an error occurs during plotting (via on.exit).

Value

Invisibly returns the entropy difference vector (the same value returned invisibly by shannon_entropy_diff_plot).

Examples

```
first_file <- system.file(
  "extdata", "example_shannon_entropy.RDS", package = "MDaRes"
)
second_file <- system.file(
  "extdata", "example_shannon_entropy_2.RDS", package = "MDaRes"
)

sh1 <- readRDS(first_file)
sh2 <- readRDS(second_file)

output_base <- tempfile("shannon-difference-")
output_file <- paste0(output_base, ".png")

do_shannon_entropy_diff_plot(sh1, sh2, output_base)
file.exists(output_file)
unlink(output_file)
```

---

do_shannon_entropy_plot

Generate Shannon Entropy Plot and Save to PNG

Description

Generates a Shannon entropy plot and saves it as a PNG file.

Usage

```
do_shannon_entropy_plot(shannon_output, output_name)
```

Arguments

shannon_output	A list of class shannon_list, as returned by shannon_entropy().
output_name	Character. The filename (without extension) for the output image.

Details

The function calls shannon_entropy_plot() to generate the plot and saves it as a PNG file.

Value

The function does not return anything but creates a PNG file.

Examples

```
shannon_file <- system.file(
  "extdata", "example_shannon_entropy.RDS", package = "MDaRes"
)
shannon_output <- readRDS(shannon_file)

output_base <- tempfile("shannon-plot-")
output_file <- paste0(output_base, ".png")

do_shannon_entropy_plot(shannon_output, output_base)
file.exists(output_file)
unlink(output_file)
```

---

entropy_visualization *Visualize Shannon Entropy on a Protein Structure*

Description

Maps fragment-based Shannon entropy values onto a protein structure by storing residue-wise entropy values in the B-factor column of a PDB file. A companion PyMOL script is generated to visualize the modified structure using cartoon putty representation and B-factor based coloring.

Usage

```
entropy_visualization(pdb_filename, chain = NULL, shannon_output)
```

Arguments

pdb_filename	Character string. Path to the input PDB or CIF file.
chain	Character string or NULL. Chain identifier to retain, for example "A". If NULL, all chains are retained.
shannon_output	A list produced by `shannon_entropy()`. It must contain the elements `entropy` and `residue_numbering`. The `residue_numbering` element must contain fragment ranges in the form `"start_end"`.

Details

Shannon entropy values are computed for structural fragments, whereas PyMOL visualizes atom-based or residue-based properties. Therefore, this function first converts fragment entropy values into residue entropy values. For each residue in the structure, the assigned entropy is calculated as the mean entropy of all fragments spanning that residue.

The resulting residue-wise entropy values are written into the B-factor column of the output PDB file. All atoms belonging to the same residue receive the same entropy value.

Two output files are generated in the same directory as the input structure:

- `<input>-bfactor.pdb`: PDB file with entropy values stored in the B-factor column.
- `<input>-bfactor.pml`: PyMOL script for cartoon putty visualization and B-factor based coloring.

Value

Invisibly returns a list with the following elements:

- `output_pdb`: Path to the generated PDB file.
- `output_pml`: Path to the generated PyMOL script.
- `entropy_per_residue`: Named numeric vector containing the entropy value assigned to each residue.

Examples

```
source_pdb <- system.file(
  "extdata", "example_calpha_ref.pdb", package = "MDaRes"
)
shannon_file <- system.file(
  "extdata", "example_shannon_entropy.RDS", package = "MDaRes"
)

working_pdb <- tempfile(fileext = ".pdb")
file.copy(source_pdb, working_pdb)
shannon_output <- readRDS(shannon_file)

result <- entropy_visualization(
  pdb_filename = working_pdb,
  chain = "A",
  shannon_output = shannon_output
)

file.exists(result$output_pdb)
file.exists(result$output_pml)

unlink(c(
  working_pdb,
  result$output_pdb,
  result$output_pml
))
```

`freq_diff_plot` *Plot Structural Frequency Differences*

Description

Generates a heatmap of per-fragment Structural Alphabet frequency differences between two `shannon_list` objects. The plotted quantity is $\Delta f = f_{first} - f_{second}$.

Usage

```
freq_diff_plot(
  first_freq,
  second_freq,
  zlim = c(-1, 1),
  n_ticks = 12,
  main = "Structural sequence profile (Deltafreq)",
  ylab = "SA Letter",
  letters = LETTERS[1:25],
 palette_fun = grDevices::colorRampPalette(c("blue", "white", "red"), space = "rgb"),
  n_col = 101,
  show_colorbar = TRUE,
  colorbar_width = 0.04,
  colorbar_gap = 0.05,
  mar = c(6, 4, 4, 6)
)
```

Arguments

first_freq	A shannon_list object containing the first frequency matrix in $freq.
second_freq	A shannon_list object containing the second frequency matrix in $freq.
zlim	Numeric vector of length 2. Limits of the color scale for Δf. Default is c(-1, 1).
n_ticks	Integer. Approximate number of x-axis residue ticks to show. Default is 12.
main	Character. Plot title.
ylab	Character. Y-axis label.
letters	Character vector of SA letters. Length must match the number of rows in the frequency matrix.
palette_fun	Function that generates the color palette.
n_col	Integer. Number of colors used for the heatmap.
show_colorbar	Logical. If TRUE, draw a right-side colorbar.
colorbar_width	Numeric. Relative width of the colorbar region.
colorbar_gap	Numeric. Gap between heatmap and colorbar.
mar	Numeric vector of length 4. Plot margins.

Details

Positive values indicate higher frequency in first_freq; negative values indicate higher frequency in second_freq. The colorbar is drawn inside an expanded x-axis region to avoid clipping when saving to image files.

Value

Invisibly returns the frequency-difference matrix.

Examples

```
first_file <- system.file(
  "extdata", "example_shannon_entropy.RDS", package = "MDaRes"
)
second_file <- system.file(
  "extdata", "example_shannon_entropy_2.RDS", package = "MDaRes"
)
sh1 <- readRDS(first_file)
sh2 <- readRDS(second_file)
freq_diff_plot(sh1, sh2)
```

`freq_plot` *Plot Frequency Profile*

Description

Generates a heatmap representing the structural sequence frequency profile obtained from Structural Alphabet encoding.

Usage

```
freq_plot(shannon_output)
```

Arguments

`shannon_output`	A list of class `shannon_list`, containing a frequency matrix and residue numbering.

Details

The frequency matrix is plotted as a heatmap where rows correspond to Structural Alphabet letters and columns correspond to encoded fragments. Residue labels on the x-axis are shown at reduced density to avoid overcrowding.

Value

A heatmap displaying the frequency of Structural Alphabet letters along the encoded sequence.

Examples

```
shannon_file <- system.file(
  "extdata", "example_shannon_entropy.RDS", package = "MDaRes"
)
shannon_output <- readRDS(shannon_file)
freq_plot(shannon_output)
```

get_res_numbering *Retrieve Residue Numbering from Encoded Trajectory*

Description

This function extracts residue numbering from the column names of an encoded trajectory. It is useful for labeling plots with meaningful residue numbers instead of generic column names.

Usage

```
get_res_numbering(encoded_trajectory)
```

Arguments

encoded_trajectory	A data frame or object of class `enc_trj` containing the encoded trajectory data. The column names of this data frame should include residue numbering information.

Value

A character vector containing cleaned residue numbering extracted from the column names.

import_SA *Import Structural Alphabet from PDB Files*

Description

Imports a structural alphabet from a set of PDB files.

Usage

```
import_SA(
  n_elements = 25,
  folder_path = normalizePath(system.file("SA", package = "MDaRes")),
  file_extension = ".pdb"
)
```

Arguments

n_elements	Integer. The number of structural alphabet elements to import.
folder_path	Character. Path to the folder containing the PDB files.
file_extension	Character. File extension of the structural alphabet files (e.g., ".pdb").

Details

This function loads `n_elements` structures from the given folder and stores them in a named list using capital letters (A-Z) as names. The **bio3d** package is used to read PDB files.

Value

A named list of PDB objects, where each element corresponds to a structural alphabet letter.

Examples

```
SA <- import_SA()
```

`MImat` *Compute Mutual Information Matrix*

Description

Computes a mutual information matrix from an **encoded trajectory**.

Usage

```
MImat(encoded_trajectory)
```

Arguments

`encoded_trajectory`
: A data frame of class `enc_trj` containing the encoded trajectory.

Value

A list of class `MI_list` containing the **mutual information matrix** and residue numbering.

Examples

```
encoding_file <- system.file(
  "extdata",
  "example_encoding.RDS",
  package = "MDaRes"
)
encoded_trj <- readRDS(encoding_file)
encoded_trj <- encoded_trj[
  , seq_len(min(10L, ncol(encoded_trj))), drop = FALSE
]
MI_output <- MImat(encoded_trj)
```

MI_plot *Plot Mutual Information Matrix*

Description

Generates a heatmap representing the mutual information between structural fragments.

Usage

```
MI_plot(MI_output)
```

Arguments

MI_output	A list of class MI_list, containing the mutual information matrix and residue numbering.

Details

The function plots the normalized mutual information matrix and labels both axes using residue numbering when available. Tick density is reduced to avoid overcrowding in large systems.

Value

A heatmap of mutual information values between structural fragments.

Examples

```
MI_file <- system.file(
  "extdata", "example_MI.RDS", package = "MDaRes"
)
MI_output <- readRDS(MI_file)
MI_plot(MI_output)
```

MI_ranking_couple *Rank Residues Based on Mutual Information*

Description

Generates rankings of residue pairs based on mutual information values.

Usage

```
MI_ranking_couple(MI_output)
```

Arguments

MI_output	A list of class MI_list containing the mutual information matrix.

Value

A data frame ranking residue pairs by mutual information value.

Examples

```
MI_file <- system.file(
  "extdata",
  "example_MI.RDS",
  package = "MDaRes"
)
MI_output <- readRDS(MI_file)
MI_ranking_couple(MI_output)
```

MI_ranking_median *Rank Mutual Information by Median and Collectivity*

Description

This function computes the median mutual information (MI) and the degree of collectivity for each fragment in a mutual information matrix. It is used to identify the most central and informative fragments based on information theory and spectral decomposition.

Usage

```
MI_ranking_median(MI_output)
```

Arguments

MI_output	An object of class MI_list, as returned by the MImat() function. It must contain a symmetric MI matrix with fragment names as column and row names.

Details

The degree of collectivity is calculated as the contribution of the top 3 eigenmodes of the mutual information matrix. This metric highlights fragments that are involved in widespread, collective information exchanges.

Value

A data frame containing the following columns:

- `start`: Starting residue index for the fragment.
- `end`: Ending residue index for the fragment.
- `median_MI`: The median mutual information value for the fragment.
- `collectivity`: A spectral-based measure of collectivity derived from the top eigenvectors of the MI matrix.

The data frame is sorted in descending order by `median_MI`.

See Also

`MImat`, `MI_ranking_couple`

Examples

```
MI_file <- system.file(
  "extdata",
  "example_MI.RDS",
  package = "MDaRes"
)
MI_output <- readRDS(MI_file)
ranked_fragments <- MI_ranking_median(MI_output)
```

---

nMImat *Compute Normalized Mutual Information Matrix*

Description

Computes the normalized **mutual information matrix** from a structural alphabet trajectory.

Usage

```
nMImat(SA_output)
```

Arguments

`SA_output`	A data frame of class `SA_trj` containing the structural alphabet trajectory.

Value

A matrix representing normalized mutual information values.

SA_encoding *Encode an MD Trajectory Using a Structural Alphabet (C++ back-end)*

Description

Encodes a molecular dynamics (MD) trajectory using a 4-residue structural alphabet (SA). For each frame and for each 4-residue fragment, the closest SA letter is assigned based on Kabsch-aligned RMSD computed in C++.

Usage

```
SA_encoding(
  SA,
  trj_name,
  top_name,
  pdb_name,
  sel_chain,
  parallel = c("none", "future"),
  workers = NULL,
  future_plan = NULL,
  verbose = TRUE
)
```

Arguments

SA	List. Structural alphabet, indexed by LETTERS (e.g. A..Y), each element containing a $xyz 4x3 numeric matrix.
trj_name	Character. Path to the trajectory file (e.g. XTC, DCD).
top_name	Character. Path to the topology file (as required by SOMMD::read.trj()).
pdb_name	Character. Path to the reference structure file (PDB or CIF).
sel_chain	Character. Chain identifier to be encoded (e.g. "A").
parallel	Character. Either "none" (serial) or "future" (parallel).
workers	Integer or NULL. Number of workers when parallel = "future". If NULL, relies on the current future plan.
future_plan	A future plan function, e.g. future::multisession(). Only used when parallel = "future".
verbose	Logical. If TRUE, prints progress messages.

Value

A data frame of class "enc_trj" with one row per frame and one column per fragment (e.g. "Frag: 1_4", "Frag: 2_5", ...), containing SA letters (A..Y).

shannon_entropy *Compute Shannon Entropy and Frequency Distribution*

Description

This function calculates the Shannon entropy and the frequency of structural alphabet letters in an encoded trajectory.

Usage

```
shannon_entropy(encoded_trajectory)
```

Arguments

encoded_trajectory	A data frame of class enc_trj, as returned by SA_encoding().

Details

Shannon entropy is calculated for each fragment in the trajectory using the frequency of different structural alphabet letters. The frequency matrix (freq) contains the normalized frequency of each letter (A-Z).

Value

A list of class shannon_list with the following components:

- freq: A 25xN matrix of frequency values, where N is the number of fragments.
- entropy: A numeric vector containing Shannon entropy values for each fragment.
- residue_numbering: A character vector representing residue numbers.

Examples

```
encoding_file <- system.file(
  "extdata",
  "example_encoding.RDS",
  package = "MDaRes"
)
encoded_trj <- readRDS(encoding_file)
shannon_output <- shannon_entropy(encoded_trj)
print(shannon_output$entropy)
```

shannon_entropy_diff_plot *Plot Shannon Entropy Difference*

Description

Plots the pointwise difference between two Shannon entropy profiles, $\Delta H = H_{first} - H_{second}$, with a zero baseline. Optionally overlays the two original entropy curves on a right-hand axis.

Usage

```
shannon_entropy_diff_plot(first_out, second_out, overlay = FALSE)
```

Arguments

first_out	A shannon_list object containing at least entropy and residue_numbering.
second_out	A shannon_list object with the same structure and length as first_out.
overlay	Logical. If TRUE, overlays the original entropy curves using a right-hand y-axis. Default is FALSE.

Details

Both inputs must be of class shannon_list and their entropy vectors must have the same length. The left y-axis is centered at 0 and scaled symmetrically to the range of ΔH. When overlay = TRUE, the original entropy curves are drawn using a right-hand axis.

Value

Invisibly returns the numeric vector ΔH used for plotting.

Examples

```
first_file <- system.file(
  "extdata", "example_shannon_entropy.RDS", package = "MDaRes"
)
second_file <- system.file(
  "extdata", "example_shannon_entropy_2.RDS", package = "MDaRes"
)
sh1 <- readRDS(first_file)
sh2 <- readRDS(second_file)
shannon_entropy_diff_plot(sh1, sh2)
shannon_entropy_diff_plot(sh1, sh2, overlay = TRUE)
```

shannon_entropy_plot *Plot Shannon Entropy*

Description

This function generates a Shannon entropy profile from a `shannon_list` object produced by `shannon_entropy()`.

Usage

```
shannon_entropy_plot(shannon_output)
```

Arguments

`shannon_output` A list of class `shannon_list`, containing entropy values and residue numbering.

Details

The function plots entropy values along the sequence and adds horizontal and vertical grid lines for readability. Residue labels on the x-axis are shown at reduced density to avoid overcrowding in large systems.

Value

A plot displaying Shannon entropy across the encoded sequence.

Examples

```
shannon_file <- system.file(
  "extdata", "example_shannon_entropy.RDS", package = "MDaRes"
)
shannon_output <- readRDS(shannon_file)
shannon_entropy_plot(shannon_output)
```

shannon_ranking *Rank Fragments by Shannon Entropy*

Description

Creates a ranking table of fragments/residue ranges ordered by Shannon entropy (descending). The function expects a `shannon_list` object containing `entropy` and `residue_numbering`.

Usage

```
shannon_ranking(shannon_output)
```

Arguments

shannon_output An object of class shannon_list with elements:

- entropy: numeric vector of Shannon entropy values (one per fragment).
- residue_numbering: character vector with residue ranges (e.g. "12_15").

Value

A data.frame with columns entropy, start, end, sorted by decreasing entropy.

Examples

```
encoding_file <- system.file(
  "extdata",
  "example_encoding.RDS",
  package = "MDaRes"
)
encoded_trj <- readRDS(encoding_file)
shannon_output <- shannon_entropy(encoded_trj)
shannon_ranking(shannon_output)
```

---

total_ranking *Merge Shannon Entropy and MI Rankings*

Description

Merges Shannon entropy and mutual information rankings into a single data frame.

Usage

```
total_ranking(shannon_ranking_df, MI_median_ranking_df)
```

Arguments

shannon_ranking_df
: Data frame containing Shannon entropy rankings.

MI_median_ranking_df
: Data frame containing mutual information median rankings.

Value

A merged ranking data frame.

Examples

```
shannon_file <- system.file(
  "extdata", "example_shannon_ranking.RDS", package = "MDaRes"
)
mi_file <- system.file(
  "extdata", "example_MI_ranking_median.RDS", package = "MDaRes"
)

shannon_ranking_df <- readRDS(shannon_file)
MI_median_ranking_df <- readRDS(mi_file)

total_ranking(shannon_ranking_df, MI_median_ranking_df)
```

Index