EVcouplings Documentation

Release 0.0.1

Thomas Hopf

Sep 16, 2018

Contents

CHAPTER 1

Alignment

1.1 evcouplings.align package

1.1.1 evcouplings.align.alignment module

Class and functions for reading, storing and manipulating multiple sequence alignments.

Authors: Thomas A. Hopf

class evcouplings.align.alignment.**Alignment**(*sequence_matrix*, *sequence_ids=None*, *annotation=None*, *alphabet='-ACDEFGHIKLMNPQRSTVWY'*)

Bases: object

Container to store and manipulate multiple sequence alignments.

Note: Important:

1. Sequence annotation currently is not transformed when selecting subsets of columns or positions (e.g. affects GR and GC lines in Stockholm alignments)
2. Sequence ranges in IDs are not adjusted when selecting subsets of positions

---

apply(*columns=None*, *sequences=None*, *func=<MagicMock name='mock.lower' id='139633675357376'>*)

Apply a function along columns and/or rows of alignment matrix, or to entire matrix.

Parameters

- **columns** (*np.array(bool) or np.array(int), optional*) – Vector containing True for each column that should be retained, False otherwise; or the indices of columns that should be selected

- **sequences** (*np.array(bool) or np.array(int), optional*) – Vector containing True for each sequence that should be retained, False otherwise; or the indices of sequences that should be selected
- **func** (*callable*) – Vectorized numpy function that will be applied to the selected subset of the alignment matrix

Returns Alignment with modified columns and sequences (this alignment maintains annotation)

Return type *Alignment*

conservation(*normalize=True*)

Calculate per-column conservation of sequence alignment based on entropy of single-column frequency distribution.

If self.set_weights() was called previously, the frequencies used to calculate site entropies will be based on reweighted sequences.

Parameters **normalize** (*bool, optional (default: True)*) – Transform column entropy to range 0 (no conservation) to 1 (fully conserved)

Returns Vector of length L with conservation scores

Return type np.array

count(*char*, *axis='pos'*, *normalize=True*)

Count occurrences of a character in the sequence alignment.

Note: The counts are raw counts not adjusted for sequence redundancy.

Parameters

- **char** (*str*) – Character which is counted
- **axis** (*{"pos", "seq"}, optional (default="pos")*) – Count along positions or sequences
- **normalize** (*bool, optional (default=True)*) – Normalize count for length of axis (i.e. relative count)

Returns Vector containing counts of char along the axis

Return type np.array

Raises ValueError – Upon invalid axis specification

frequencies

Returns/calculates single-site frequencies of symbols in alignment. Also sets self._frequencies member variable for later reuse.

Previously calculated sequence weights using self.set_weights() will be used to adjust frequency counts; otherwise, each sequence will contribute with equal weight.

Returns Reference to self._frequencies

Return type np.array

classmethod from_dict(*sequences*, ***kwargs*)

Construct an alignment object from a dictionary with sequence IDs as keys and aligned sequences as values.

Parameters **sequences** (*dict-like*) – Dictionary with pairs of sequence ID (key) and aligned sequence (value)

Returns initialized alignment

Return type *Alignment*

classmethod from_file(*fileobj*, *format='fasta'*, *a3m_inserts='first'*, *raise_hmmer_prefixes=True*, ***kwargs*)

Construct an alignment object by reading in an alignment file.

Parameters

- **fileobj** (*file-like obj*) – Alignment to be read in
- **format** (*{"fasta", "stockholm", "a3m"}*) – Format of input alignment
- **a3m_inserts** (*{"first", "delete"}, optional (default: "first")*) – Strategy to deal with inserts in a3m alignment files (see read_a3m documentation for details)
- **raise_hmmer_prefixes** (*bool, optional (default: True)*) – HMMER adds number prefixes to sequence identifiers in Stockholm files if identifiers are not unique. If True, the parser will raise an exception if a Stockholm alignment has such prefixes.
- ****kwargs** – Additional arguments to be passed to class constructor

Returns Parsed alignment

Return type *Alignment*

Raises ValueError – For invalid alignments or alignment formats

identities_to(*seq*, *normalize=True*)

Calculate sequence identity between sequence and all sequences in the alignment.

seq [np.array, list-like, or str] Sequence for comparison

normalize [bool, optional (default: True)] Calculate relative identity between 0 and 1 by normalizing with length of alignment

lowercase_columns(*columns*)

Change a subset of columns to lowercase character and replace "-" gaps with "." gaps, e.g. to exclude them from EC calculations

Parameters **columns** (*numpy index array*) – Subset of columns to make lowercase

Returns Alignment with lowercase columns

Return type *Alignment*

pair_frequencies

Returns/calculates pairwise frequencies of symbols in alignment. Also sets self._pair_frequencies member variable for later reuse.

Previously calculated sequence weights using self.set_weights() will be used to adjust frequency counts; otherwise, each sequence will contribute with equal weight.

Returns Reference to self._pair_frequencies

Return type np.array

replace(*original*, *replacement*, *columns=None*, *sequences=None*)

Replace character with another in full matrix or subset of columns/sequences.

Parameters

- **original** (*char*) – Character that should be replaced
- **replacement** (*char*) – Replacement character
- **columns** (*numpy index array*) – See self.apply for explanation
- **sequences** (*numpy index array*) – See self.apply for explanation

Returns Alignment with replaced characters

Return type *Alignment*

select (*columns=None*, *sequences=None*)
Create a sub-alignment that contains a subset of sequences and/or columns.

Note: This does currently not adjust the indices of the sequences. Annotation in the original alignment will be lost and not passed on to the new object.

Parameters

- **columns** (*np.array(bool) or np.array(int), optional*) – Vector containing True for each column that should be retained, False otherwise; or the indices of columns that should be selected
- **sequences** (*np.array(bool) or np.array(int), optional*) – Vector containing True for each sequence that should be retained, False otherwise; or the indices of sequences that should be selected

Returns Alignment with selected columns and sequences (note this alignment looses annotation)

Return type *Alignment*

set_weights (*identity_threshold=0.8*)
Calculate weights for sequences in alignment by clustering all sequences with sequence identity greater or equal to the given threshold.

Note: This method sets self.weights. After this method was called, methods/attributes such as self.frequencies or self.conservation() will make use of sequence weights.

Note: (Implementation: cannot use property here since we need identity threshold as a parameter. . . .)

Parameters **identity_threshold** (*float, optional (default: 0.8)*) – Sequence identity threshold

write (*fileobj*, *format='fasta'*, *width=80*)
Write an alignment to a file.

Parameters

- **fileobj** (*file-like object*) – File to which alignment is saved
- **format** (*{"fasta", "aln", "a3m"}*) – Output format for alignment
- **width** (*int*) – Column width for fasta alignment

Raises ValueError – Upon invalid file format specification

class evcouplings.align.alignment.**StockholmAlignment**(*seqs*, *gf*, *gc*, *gs*, *gr*)
Bases: tuple

gc
Alias for field number 2

gf
Alias for field number 1

gr
Alias for field number 4

gs
Alias for field number 3

seqs
Alias for field number 0

evcouplings.align.alignment.**detect_format**(*fileobj*)
Detect if an alignment file is in FASTA or Stockholm format.

Parameters **fileobj** (*file-like obj*) – Alignment file for which to detect format

Returns **format** – Format of alignment, None if not detectable

Return type {"fasta", "stockholm", None}

evcouplings.align.alignment.**map_from_alphabet**(*alphabet='-ACDEFGHIKLMNPQRSTVWY'*, *default='-'*)
Creates a mapping dictionary from a given alphabet.

Parameters

- **alphabet** (*str*) – Alphabet for remapping. Elements will be remapped according to alphabet starting from 0
- **default** (*Elements in matrix that are not*) – contained in alphabet will be treated as this character

Raises ValueError – For invalid default character

evcouplings.align.alignment.**map_matrix**(*matrix*, *map_*)
Map elements in a numpy array using alphabet

Parameters

- **matrix** (*np.array*) – Matrix that should be remapped
- **map** (*defaultdict*) – Map that will be applied to matrix elements

Returns Remapped matrix

Return type np.array

evcouplings.align.alignment.**parse_header**(*header*)
Extract ID of the (overall) sequence and the sequence range fromat a sequence header of the form sequenceID/start-end.

If the header contains any additional information after the first whitespace character (e.g. sequence annotation), it will be discarded before parsing. If there is no sequence range, only the id (part of string before whitespace) will be returned but no range.

Parameters **header** (*str*) – Sequence header

Returns

- **seq_id** (*str*) – Sequence identifier
- **start** (*int*) – Start of sequence range. Will be None if it cannot be extracted from the header.
- **stop** (*int*) – End of sequence range. Will be None if it cannot be extracted from the header.

evcouplings.align.alignment.**read_a3m**(*fileobj*, *inserts='first'*)

Read an alignment in compressed a3m format and expand into a2m format.

Note: this function is currently not able to keep inserts in all the sequences

..todo:

```
implement this
```

Parameters

- **fileobj** (*file-like object*) – A3M alignment file
- **inserts** (*{"first", "delete"}*) – Keep inserts in first sequence, or delete any insert column and keep only match state columns.

Returns Sequences in alignment (key: ID, value: sequence), in order they appeared in input file

Return type OrderedDict

Raises ValueError – Upon invalid choice of insert strategy

evcouplings.align.alignment.**read_fasta**(*fileobj*)

Generator function to read a FASTA-format file (includes aligned FASTA, A2M, A3M formats)

Parameters **fileobj** (*file-like object*) – FASTA alignment file

Returns Returns tuples of (sequence ID, sequence)

Return type generator of (str, str) tuples

evcouplings.align.alignment.**read_stockholm**(*fileobj*, *read_annotation=False*, *raise_hmmer_prefixes=True*)

Generator function to read Stockholm format alignment file (e.g. from hmmer).

Note: Generator iterates over different alignments in the same file (but not over individual sequences, which makes little sense due to wrapped Stockholm alignments).

Parameters

- **fileobj** (*file-like object*) – Stockholm alignment file
- **read_annotation** (*bool, optional (default=False)*) – Read annotation columns from alignment
- **raise_hmmer_prefixes** (*bool, optional (default: True)*) – HMMER adds number prefixes to sequence identifiers if identifiers are not unique. If True, the parser will raise an exception if the alignment has such prefixes.

Returns namedtuple with the following fields: seqs, gf, gc, gs, gr (all DefaultOrderedDict)

Return type *StockholmAlignment*

Raises ValueError

evcouplings.align.alignment.**sequences_to_matrix**(*sequences*)
Transforms a list of sequences into a numpy array.

Parameters **sequences** (*list-like (str)*) – List of strings containing aligned sequences

Returns 2D array containing sequence alignment (first axis: sequences, second axis: columns)

Return type numpy.array

evcouplings.align.alignment.**write_a3m**(*sequences*, *fileobj*, *insert_gap='.'*, *width=80*)
Write a list of IDs/sequences to a FASTA-format file

Parameters

- **sequences** (*Iterable*) – Iterator returning tuples(str, str), where first value is header/ID and second the sequence
- **fileobj** (*file-like obj*) – File to which alignment will be written

evcouplings.align.alignment.**write_aln**(*sequences*, *fileobj*, *width=80*)
Write a list of IDs/sequences to a ALN-format file

Currently, the file will not contain headers but simply a block matrix of the alignment

Parameters

- **sequences** (*Iterable*) – Iterator returning tuples(str, str), where first value is header/ID and second the sequence
- **fileobj** (*file-like obj*) – File to which alignment will be written

evcouplings.align.alignment.**write_fasta**(*sequences*, *fileobj*, *width=80*)
Write a list of IDs/sequences to a FASTA-format file

Parameters

- **sequences** (*Iterable*) – Iterator returning tuples(str, str), where first value is header/ID and second the sequence
- **fileobj** (*file-like obj*) – File to which alignment will be written

1.1.2 evcouplings.align.pfam module

Code for identifying Pfam domains and mapping Pfam alignments and ECs into target sequence mode.

Authors: Thomas A. Hopf

Todo:

1. Write code to create list of family sizes
2. Implement alignments against Pfam-HMM so precomputed results can be reused in focus mode

---

evcouplings.align.pfam.**create_family_size_table**(*full_pfam_file*, *outfile=None*)
Parse family size table from Pfam flat file (ftp://ftp.ebi.ac.uk/pub/databases/Pfam/current_release/Pfam-A.full.gz)

Parameters

- **full_pfam_file** (*str*) – Path to the pfam file (gzip).
- **outfile** (*str, optional (default: None)*) – Save the parsed table to this file as a csv file.

Returns Parsed Pfam table.

Return type pd.DataFrame

evcouplings.align.pfam.**pfam_hits**(*query_file*, *hmm_database*, *prefix*, *clan_table_file*, *size_table_file*, *resolve_overlaps=True*, *hmmscan_binary='hmmscan'*)

Identify hits of Pfam HMMs in a set of sequences.

Parameters

- **query_file** (*str*) – File containing query sequence(s)
- **hmm_database** (*str*) – File containing HMM database (Pfam-A.hmm, with hmmpress applied)
- **prefix** (*str*) – Prefix path for output files. Folder structure in the prefix will be created if not existing.
- **clan_table_file** (*str*) – File with table linking Pfam families to clans (Pfam-A.clans.tsv). Set to None if not available, but resolve_overlaps cannot be True in that case.
- **size_table_file** (*str*) – File with table of family sizes. Create using create_family_size_table(). Set to None if not available.
- **resolve_overlaps** (*bool*) – Resolve overlapping hits by families from the same clan. Only possible if clan_table_file is given.
- **hmmscan_binary** (*str (default: "hmmscan")*) – Path to hmmscan binary (put in PATH for default to work)

Returns Pfam hit table

Return type pd.DataFrame

evcouplings.align.pfam.**remove_clan_overlaps**(*pfam_table*)

Remove overlapping Pfam hits from same Pfam clan (equivalent of PfamScan.pl). Currently only allows to remove overlaps by domain bitscore.

Todo: is bitscore the most sensible choice if different length hits?

Parameters **pfam_table** (*pd.DataFrame*) – Pfam hit table as generated by pfam_hits() function (must contain Pfam clan annotation).

Returns Pfam hit table with lower-scoring overlaps removed

Return type pd.DataFrame

1.1.3 evcouplings.align.protocol module

Protein sequence alignment creation protocols/workflows.

Authors: Thomas A. Hopf Anna G. Green - complex protocol, hmm_build_and_search Chan Kang - hmm_build_and_search

evcouplings.align.protocol.**complex**(***kwargs*)

Protocol:

Run monomer alignment protocol and postprocess it for EVcomplex calculations

Parameters **kwargs arguments** (*Mandatory*) – See list below in code where calling check_required

Returns

outcfg – Output configuration of the alignment protocol, and the following additional field:

genome_location_file [path to file containing] the genomic locations for CDs's corresponding to identifiers in the alignment.

Return type dict

evcouplings.align.protocol.**cut_sequence**(*sequence*, *sequence_id*, *region=None*, *first_index=None*, *out_file=None*)

Cut a given sequence to sub-range and save it in a file

Parameters

- **sequence** (*str*) – Full sequence that will be cut
- **sequence_id** (*str*) – Identifier of sequence, used to construct header in output file
- **region** (*tuple(int, int), optional (default: None)*) – Region that will be cut out of full sequence. If None, full sequence will be returned.
- **first_index** (*int, optional (default: None)*) – Define index of first position in sequence. Will be set to 1 if None.
- **out_file** (*str, optional (default: None)*) – Save sequence in a FASTA file (header: >sequence_id/start_region-end_region)

Returns

- *str* – Subsequence contained in region
- *tuple(int, int)* – Region. If no input region is given, this will be (1, len(sequence)); otherwise, the input region is returned.

Raises InvalidParameterError – Upon invalid region specification (violating boundaries of sequence)

evcouplings.align.protocol.**describe_coverage**(*alignment*, *prefix*, *first_index*, *minimum_column_coverage*)

Produce "classical" buildali coverage statistics, i.e. number of sequences, how many residues have too many gaps, etc.

Only to be applied to alignments focused around the target sequence.

Parameters

- **alignment** (Alignment) – Alignment for which coverage statistics will be calculated
- **prefix** (*str*) – Prefix of alignment file that will be stored as identifier in table
- **first_index** (*int*) – Sequence index of first position of target sequence
- **minimum_column_coverage** (*Iterable(float) or float*) – Minimum column coverage threshold(s) that will be tested (creating one row for each threshold in output table).

Note: int values given to this function instead of a float will be divided by 100 to create the corresponding floating point representation. This parameter is 1.0 - maximum fraction of gaps per column.

Returns Table with coverage statistics for different gap thresholds

Return type pd.DataFrame

evcouplings.align.protocol.**describe_frequencies**(*alignment*, *first_index*, *target_seq_index=None*)

Get parameters of alignment such as gaps, coverage, conservation and summarize.

Parameters

- **alignment** (Alignment) – Alignment for which description statistics will be calculated
- **first_index** (*int*) – Sequence index of first residue in target sequence
- **target_seq_index** (*int, optional (default: None)*) – If given, will add the symbol in the target sequence into a separate column of the output table

Returns Table detailing conservation and symbol frequencies for all positions in the alignment

Return type pandas.DataFrame

evcouplings.align.protocol.**describe_seq_identities**(*alignment*, *target_seq_index=0*)

Calculate sequence identities of any sequence to target sequence and create result dataframe.

Parameters **alignment** (Alignment) – Alignment for which description statistics will be calculated

Returns Table giving the identity to target sequence for each sequence in alignment (in order of occurrence)

Return type pandas.DataFrame

evcouplings.align.protocol.**existing**(***kwargs*)

Protocol:

Use external sequence alignment and extract all relevant information from there (e.g. sequence, region, etc.), then apply gap & fragment filtering as usual

Parameters **kwargs arguments** (*Mandatory*) – See list below in code where calling check_required

Returns

outcfg – Output configuration of the pipeline, including the following fields:

- sequence_id (passed through from input)
- alignment_file
- raw_focus_alignment_file
- statistics_file
- sequence_file
- first_index
- target_sequence_file
- annotation_file (None)
- frequencies_file
- identities_file
- focus_mode
- focus_sequence

- segments

Return type dict

evcouplings.align.protocol.**extract_header_annotation**(*alignment*, *from_annotation=True*)

Extract Uniprot/Uniref sequence annotation from Stockholm file (as output by jackhmmer). This function may not work for other formats.

Parameters

- **alignment** (Alignment) – Multiple sequence alignment object
- **from_annotation** (*bool, optional (default: True)*) – Use annotation line (in Stockholm file) rather than sequence ID line (e.g. in FASTA file)

Returns Table containing all annotation (one row per sequence in alignment, in order of occurrence)

Return type pandas.DataFrame

evcouplings.align.protocol.**fetch_sequence**(*sequence_id*, *sequence_file*, *sequence_download_url*, *out_file*)

Fetch sequence either from database based on identifier, or from input sequence file.

Parameters

- **sequence_id** (*str*) – Identifier of sequence that should be retrieved
- **sequence_file** (*str*) – File containing sequence. If None, sqeuence will be downloaded from sequence_download_url
- **sequence_download_url** (*str*) – URL from which to download missing sequence. Must contain "{}" at the position where sequence ID will be inserted into download URL (using str.format).
- **out_file** (*str*) – Output file in which sequence will be stored, if sequence_file is not existing.

Returns

- *str* – Path of file with stored sequence (can be sequence_file or out_file)
- *tuple (str, str)* – Identifier of sequence as stored in file, and sequence

evcouplings.align.protocol.**hmmbuild_and_search**(***kwargs*)

Protocol:

Build HMM from sequence alignment using hmmbuild and search against a sequence database using hmmsearch.

Parameters **kwargs arguments** (*Mandatory*) – See list below in code where calling check_required

Returns

outcfg – Output configuration of the protocol, including the following fields:

- target_sequence_file
- sequence_file
- raw_alignment_file
- hittable_file
- focus_mode
- focus_sequence

- segments

Return type dict

evcouplings.align.protocol.**jackhmmer_search**(***kwargs*)

Protocol:

Iterative jackhmmer search against a sequence database.

Parameters **kwargs arguments** (*Mandatory*) – See list below in code where calling check_required

:param .. todo::: explain meaning of parameters in detail.

Returns

outcfg – Output configuration of the protocol, including the following fields:

- sequence_id (passed through from input)
- first_index (passed through from input)
- target_sequence_file
- sequence_file
- raw_alignment_file
- hittable_file
- focus_mode
- focus_sequence
- segments

Return type dict

evcouplings.align.protocol.**modify_alignment**(*focus_ali*, *target_seq_index*, *target_seq_id*, *region_start*, ***kwargs*)

Apply pairwise identity filtering, fragment filtering, and exclusion of columns with too many gaps to a sequence alignment. Also generates files describing properties of the alignment such as frequency distributions, conservation, and "old-style" alignment statistics files.

Note: assumes focus alignment (otherwise unprocessed) as input.

Todo: come up with something more clever to filter fragments than fixed width (e.g. use 95% quantile of length distribution as reference point)

Parameters

- **focus_ali** (Alignment) – Focus-mode input alignment
- **target_seq_index** (*int*) – Index of target sequence in alignment
- **target_seq_id** (*str*) – Identifier of target sequence (without range)
- **region_start** (*int*) – Index of first sequence position in target sequence
- **kwargs** (*See required arguments in source code*) –

Returns

- **outcfg** (*Dict*) – File products generated by the function:

  - alignment_file
  - statistics_file
  - frequencies_file
  - identities_file
  - raw_focus_alignment_file
- **ali** (*Alignment*) – Final processed alignment

evcouplings.align.protocol.**run**(***kwargs*)

Run alignment protocol to generate multiple sequence alignment from input sequence.

Parameters

- **kwargs arguments** (*Mandatory*) – protocol: Alignment protocol to run prefix: Output prefix for all generated files
- **Optional** –

Returns

- *Alignment*
- *Dictionary with results of stage in following fields (in brackets - not returned by all protocols)* –
  - alignment_file
  - [raw_alignment_file]
  - statistics_file
  - target_sequence_file
  - sequence_file
  - [annotation_file]
  - frequencies_file
  - identities_file
  - [hittable_file]
  - focus_mode
  - focus_sequence
  - segments

evcouplings.align.protocol.**search_thresholds**(*use_bitscores*, *seq_threshold*, *domain_threshold*, *seq_len*)

Set homology search inclusion parameters.

HMMER hits get included in the HMM according to a two-step rule

1. sequence passes sequence-level treshold
2. domain passes domain-level threshold

Therefore, search thresholds are set based on the following logic:

1. If only sequence threshold is given, a MissingParameterException is raised
2. If only bitscore threshold is given, sequence threshold is set to the same
3. If both thresholds are given, they are according to defined values

Valid inputs for bitscore thresholds:

1. int or str: taken as absolute score threshold
2. float: taken as relative threshold (absolute threshold derived by

multiplication with domain length)

Valid inputs for integer thresholds:

1. int: Used as negative exponent, threshold will be set to 1E-<exponent>
2. float or str: Interpreted literally

Parameters

- **use_bitscores** (*bool*) – Use bitscore threshold instead of E-value threshold
- **domain_threshold** (*str or int or float*) – Domain-level threshold. See rules above.
- **seq_threshold** (*str or int or float*) – Sequence-level threshold. See rules above.
- **seq_len** (*int*) – Length of sequence. Used to calculate absolute bitscore threshold for relative bitscore thresholds.

Returns Sequence- and domain-level thresholds ready to be fed into HMMER

Return type tuple (str, str)

evcouplings.align.protocol.**standard**(***kwargs*)

Protocol:

Standard buildali4 workflow (run iterative jackhmmer search against sequence database, than determine which sequences and columns to include in the calculation based on coverage and maximum gap thresholds).

Parameters **kwargs arguments** (*Mandatory*) – See list below in code where calling check_required

Returns

- **outcfg** (*dict*) – Output configuration of the pipeline, including the following fields:
  - sequence_id (passed through from input)
  - first_index (passed through from input)
  - alignment_file
  - raw_alignment_file
  - raw_focus_alignment_file
  - statistics_file
  - target_sequence_file
  - sequence_file
  - annotation_file
  - frequencies_file
  - identities_file
  - hittable_file

  - focus_mode
  - focus_sequence
  - segments
- **ali** (*Alignment*) – Final sequence alignment

1.1.4 evcouplings.align.tools module

Wrappers for running external sequence alignment tools

Authors: Thomas A. Hopf Anna G. Green - run_hmmbuild, run_hmmsearch Chan Kang - run_hmmbuild, run_hmmsearch

class evcouplings.align.tools.**HmmbuildResult**(*prefix*, *hmmfile*, *output*)
Bases: tuple

hmmfile
Alias for field number 1

output
Alias for field number 2

prefix
Alias for field number 0

class evcouplings.align.tools.**HmmscanResult**(*prefix*, *output*, *tblout*, *domtblout*, *pfamtblout*)
Bases: tuple

domtblout
Alias for field number 3

output
Alias for field number 1

pfamtblout
Alias for field number 4

prefix
Alias for field number 0

tblout
Alias for field number 2

class evcouplings.align.tools.**HmmsearchResult**(*prefix*, *alignment*, *output*, *tblout*, *domtblout*)
Bases: tuple

alignment
Alias for field number 1

domtblout
Alias for field number 4

output
Alias for field number 2

prefix
Alias for field number 0

tblout
Alias for field number 3

class evcouplings.align.tools.**JackhmmerResult**(*prefix*, *alignment*, *output*, *tblout*, *domtblout*)

Bases: tuple

alignment
Alias for field number 1

domtblout
Alias for field number 4

output
Alias for field number 2

prefix
Alias for field number 0

tblout
Alias for field number 3

evcouplings.align.tools.**read_hmmer_domtbl**(*filename*)
Read a HMMER domtbl file into DataFrame.

Parameters **filename** (*str*) – Path of domtbl file

Returns DataFrame with parsed domtbl

Return type pd.DataFrame

evcouplings.align.tools.**read_hmmer_tbl**(*filename*)
Read a HMMER tbl file into DataFrame.

Parameters **filename** (*str*) – Path of tbl file

Returns DataFrame with parsed tbl

Return type pd.DataFrame

evcouplings.align.tools.**run_hhfilter**(*input_file*, *output_file*, *threshold=95*, *columns='a2m'*, *binary='hhfilter'*)
Redundancy-reduce a sequence alignment using hhfilter from the HHsuite alignment suite.

Parameters

- **input_file** (*str*) – Path to input alignment in A2M/FASTA format
- **output_file** (*str*) – Path to output alignment (will be in A3M format)
- **threshold** (*int, optional (default: 95)*) – Sequence identity threshold for maximum pairwise identity (between 0 and 100)
- **columns** (*{"first", "a2m"}, optional (default: "a2m")*) – Definition of match columns (based on first sequence or upper-case columns (a2m))
- **binary** (*str*) – Path to hhfilter binary

Returns output_file

Return type str

Raises

- ResourceError – If output alignment is non-existent/empty
- ValueError – Upon invalid value of columns parameter

evcouplings.align.tools.**run_hmmbuild**(*alignment_file*, *prefix*, *cpu=None*, *stdout_redirect=None*, *symfrac=None*, *binary='hmmbuild'*)

Profile HMM construction from multiple sequence alignments Refer to HMMER documentation for details.

http://eddylab.org/software/hmmer3/3.1b2/Userguide.pdf

Parameters

- **alignment_file** (*str*) – File containing the multiple sequence alignment. Can be in Stockholm, a2m, or clustal formats, or any other format recognized by hmmer. Please note that ALL POSITIONS above the symfrac cutoff will be used in HMM construction (if the alignment contains columns that are insertions relative to the query sequence, this may be problematic for structure comparison)
- **prefix** (*str*) – Prefix path for output files. Folder structure in the prefix will be created if not existing.
- **cpu** (*int, optional (default: None)*) – Number of CPUs to use for search. Uses all if None.
- **stdout_redirect** (*str, optional (default: None)*) – Redirect bulky stdout instead of storing with rest of results (use "/dev/null" to dispose)
- **symfrac** (*float, optional (default: None)*) – range 0.0 - 1.0, HMMbuild will use columns with > symfrac percent gaps to construct the HMM. If None provided, HMMbuild internal default is 0.5. (Note: this is calculated after their internal sequence weighting is calculated)
- **binary** (*str (default: "hmmbuild")*) – Path to jackhmmer binary (put in PATH for default to work)

Returns namedtuple with fields corresponding to the different output files (prefix, alignment, output, tblout, domtblout)

Return type *HmmbuildResult*

Raises ExternalToolError, ResourceError

evcouplings.align.tools.**run_hmmscan**(*query*, *database*, *prefix*, *use_model_threshold=True*, *threshold_type='cut_ga'*, *use_bitscores=True*, *domain_threshold=None*, *seq_threshold=None*, *nobias=False*, *cpu=None*, *stdout_redirect=None*, *binary='hmmscan'*)

Run hmmscan of HMMs in database against sequences in query to identify matches of these HMMs. Refer to HMMER Userguide for explanation of these parameters.

Parameters

- **query** (*str*) – File containing query sequence(s)
- **database** (*str*) – File containing HMM database (prepared with hmmpress)
- **prefix** (*str*) – Prefix path for output files. Folder structure in the prefix will be created if not existing.
- **use_model_threshold** (*bool (default: True)*) – Use model-specific inclusion thresholds from HMM database rather than global bitscore/E-value thresholds (use_bitscores, domain_threshold and seq_threshold are overriden by this flag).
- **threshold-type** (*{"cut_ga", "cut_nc", "cut_tc"} (default: "cut_ga")*) – Use gathering (default), noise or trusted cutoff to define scan hits. Please refer to HMMER manual for details.

- **use_bitscores** (*bool*) – Use bitscore inclusion thresholds rather than E-values. Overriden by use_model_threshold flag.
- **domain_threshold** (*int or float or str*) – Inclusion threshold applied on the domain level (e.g. "1E-03" or 0.001 or 50)
- **seq_threshold** (*int or float or str*) – Inclusion threshold applied on the sequence level (e.g. "1E-03" or 0.001 or 50)
- **nobias** (*bool, optional (default: False)*) – Turn of bias correction
- **cpu** (*int, optional (default: None)*) – Number of CPUs to use for search. Uses all if None.
- **stdout_redirect** (*str, optional (default: None)*) – Redirect bulky stdout instead of storing with rest of results (use "/dev/null" to dispose)
- **binary** (*str (default: "hmmscan")*) – Path to hmmscan binary (put in PATH for default to work)

Returns namedtuple with fields corresponding to the different output files (prefix, output, tblout, domtblout, pfamtblout)

Return type *HmmscanResult*

Raises ExternalToolError, ResourceError

evcouplings.align.tools.**run_hmmsearch**(*hmmfile*, *database*, *prefix*, *use_bitscores*, *domain_threshold*, *seq_threshold*, *nobias=False*, *cpu=None*, *stdout_redirect=None*, *binary='hmmsearch'*)

Search profile(s) against a sequence database. Refer to HMMER documentation for details.

http://eddylab.org/software/hmmer3/3.1b2/Userguide.pdf

Parameters

- **hmmfile** (*str*) – File containing the profile(s)
- **database** (*str*) – File containing sequence database
- **prefix** (*str*) – Prefix path for output files. Folder structure in the prefix will be created if not existing.
- **use_bitscores** (*bool*) – Use bitscore inclusion thresholds rather than E-values.
- **domain_threshold** (*int or float or str*) – Inclusion threshold applied on the domain level (e.g. "1E-03" or 0.001 or 50)
- **seq_threshold** (*int or float or str*) – Inclusion threshold applied on the sequence level (e.g. "1E-03" or 0.001 or 50)
- **nobias** (*bool, optional (default: False)*) – Turn of bias correction
- **cpu** (*int, optional (default: None)*) – Number of CPUs to use for search. Uses all if None.
- **stdout_redirect** (*str, optional (default: None)*) – Redirect bulky stdout instead of storing with rest of results (use "/dev/null" to dispose)
- **binary** (*str (default: "hmmsearch")*) – Path to jackhmmer binary (put in PATH for default to work)

Returns namedtuple with fields corresponding to the different output files (prefix, alignment, output, tblout, domtblout)

Return type *HmmsearchResult*

Raises ExternalToolError, ResourceError

evcouplings.align.tools.**run_jackhmmer**(*query*, *database*, *prefix*, *use_bitscores*, *domain_threshold*, *seq_threshold*, *iterations=5*, *nobias=False*, *cpu=None*, *stdout_redirect=None*, *checkpoints_hmm=False*, *checkpoints_ali=False*, *binary='jackhmmer'*)

Run jackhmmer sequence search against target database. Refer to HMMER Userguide for explanation of these parameters.

Parameters

- **query** (*str*) – File containing query sequence
- **database** (*str*) – File containing sequence database
- **prefix** (*str*) – Prefix path for output files. Folder structure in the prefix will be created if not existing.
- **use_bitscores** (*bool*) – Use bitscore inclusion thresholds rather than E-values.
- **domain_threshold** (*int or float or str*) – Inclusion threshold applied on the domain level (e.g. "1E-03" or 0.001 or 50)
- **seq_threshold** (*int or float or str*) – Inclusion threshold applied on the sequence level (e.g. "1E-03" or 0.001 or 50)
- **iterations** (*int*) – number of jackhmmer search iterations
- **nobias** (*bool, optional (default: False)*) – Turn of bias correction
- **cpu** (*int, optional (default: None)*) – Number of CPUs to use for search. Uses all if None.
- **stdout_redirect** (*str, optional (default: None)*) – Redirect bulky stdout instead of storing with rest of results (use "/dev/null" to dispose)
- **checkpoints_hmm** (*bool, optional (default: False)*) – Store checkpoint HMMs to prefix.<iter>.hmm
- **checkpoints_ali** (*bool, optional (default: False)*) – Store checkpoint alignments to prefix.<iter>.sto
- **binary** (*str (default: "jackhmmer")*) – Path to jackhmmer binary (put in PATH for default to work)

Returns namedtuple with fields corresponding to the different output files (prefix, alignment, output, tblout, domtblout)

Return type *JackhmmerResult*

Raises ExternalToolError, ResourceError

CHAPTER 2

Couplings Analysis

2.1 evcouplings.compare package

2.1.1 evcouplings.compare.distances module

Distance calculations on PDB 3D coordinates

Authors: Thomas A. Hopf Anna G. Green (remap_complex_chains)

class evcouplings.compare.distances.**DistanceMap**(*residues_i*, *residues_j*, *dist_matrix*, *symmetric*)

Bases: object

Compute, store and accesss pairwise residue distances in PDB 3D structures

classmethod aggregate(**matrices*, *intersect=False*, *agg_func=<MagicMock id='139633674650624'>*)

Aggregate with other distance map(s). Secondary structure will be aggregated by assigning the most frequent state across all distance matrices; if there are equal counts, H (helix) will be chosen over E (strand) over C (coil).

Parameters

- ***matrices** (DistanceMap) – **args*-style list of DistanceMaps that will be aggregated.

  Note: The id column of each axis may only contain numeric residue ids (and no characters such as insertion codes)

- **intersect** (*bool, optional (default: False)*) – If True, intersect indices of the given distance maps. Otherwise, union of indices will be used.
- **agg_func** (*function (default: numpy.nanmin)*) – Function that will be used to aggregate distance matrices. Needs to take a parameter "axis" to aggregate over all matrices.

Returns Aggregated distance map

Return type *DistanceMap*

Raises `ValueError` – If residue identifiers are not numeric, or if intersect is True, but positions on axis do not overlap.

contacts(*max_dist=5.0*, *min_dist=None*)
Return list of pairs below distance threshold

Parameters

- **max_dist** (*float, optional (default: 5.0)*) – Maximum distance for any pair to be considered a contact
- **min_dist** (*float, optional (default: None)*) – Minimum distance of any pair to be returned (may be useful if extracting different distance ranges from matrix). Distance has to be > min_dist, (not >=).

Returns

contacts – Table with residue-residue contacts, with the following columns:

1. id_i: identifier of residue in chain i
2. id_j: identifier of residue in chain j
3. dist: pair distance

Return type pandas.DataFrame

dist(*i*, *j*, *raise_na=True*)
Return distance of residue pair

Parameters

- **i** (*int or str*) – Identifier of position on first axis
- **j** (*int or str*) – Identifier of position on second axis
- **raise_na** (*bool, optional (default: True)*) – Raise error if i or j is not contained in either axis. If False, returns np.nan for undefined entries.

Returns Distance of pair (i, j). If raise_na is False and identifiers are not valid, distance will be np.nan

Return type np.float

Raises `KeyError` – If index i or j is not a valid identifier for respective chain

classmethod from_coords(*chain_i*, *chain_j=None*)
Compute distance matrix from PDB chain coordinates.

Parameters

- **chain_i** (`Chain`) – PDB chain to be used for first axis of matrix
- **chain_j** (`Chain`, *optional (default: None)*) – PDB chain to be used for second axis of matrix. If not given, will be set to chain_i, resulting in a symmetric distance matrix

Returns Distance map computed from given coordinates

Return type *DistanceMap*

classmethod from_file(*filename*)
Load existing distance map from file

Parameters **filename** (*str*) – Path to distance map file

Returns Loaded distance map

Return type *DistanceMap*

to_file(*filename*)
Store distance map in file

Parameters **filename** (*str*) – Prefix of distance map files (will create .csv and .npy file)

transpose()
Transpose distance map (i.e. swap axes)

Returns Transposed copy of distance map

Return type *DistanceMap*

evcouplings.compare.distances.**inter_dists**(*sifts_result_i*, *sifts_result_j*, *structures=None*, *atom_filter=None*, *intersect=False*, *output_prefix=None*, *model=0*, *raise_missing=True*)
Compute inter-chain distances (between different entities) in PDB file. Resulting distance map is typically not symmetric, with either axis corresponding to either chain. Inter-distances are calculated on all combinations of chains that have the same PDB id in sifts_result_i and sifts_result_j.

Parameters

- **sifts_result_i** (SIFTSResult) – Input structures and mapping to use for first axis of computed distance map
- **sifts_result_j** (SIFTSResult) – Input structures and mapping to use for second axis of computed distance map
- **structures** (*str or dict, optional (default: None)*) –
  - If str: Load structures from directory this string points to. Missing structures will be fetched from web.
  - If dict: dictionary with lower-case PDB ids as keys and PDB objects as values. This dictionary has to contain all necessary structures, missing ones will not be fetched. This dictionary can be created using pdb.load_structures.
- **atom_filter** (*str, optional (default: None)*) – Filter coordinates to contain only these atoms. E.g. set to "CA" to compute C_alpha - C_alpha distances instead of minimum atom distance over all atoms in both residues.
- **intersect** (*bool, optional (default: False)*) – If True, intersect indices of the given distance maps. Otherwise, union of indices will be used.
- **output_prefix** (*str, optional (default: None)*) – If given, save individual and final contact maps to files prefixed with this string. The appended file suffixes map to row index in sifts_results.hits
- **model** (*int, optional (default: 0)*) – Index of model in PDB structure that should be used
- **raise_missing** (*bool, optional (default: True)*) – Raise a ResourceError if any of the input structures can not be loaded; otherwise, ignore missing entries.

Returns Computed aggregated distance map across all input structures

Return type *DistanceMap*

Raises

- ValueError – If sifts_result_i or sifts_result_j is empty (no structure hits)
- ResourceError – If any structure could not be loaded and raise_missing is True

evcouplings.compare.distances.**intra_dists**(*sifts_result*, *structures=None*, *atom_filter=None*, *intersect=False*, *output_prefix=None*, *model=0*, *raise_missing=True*)

Compute intra-chain distances in PDB files.

Parameters

- **sifts_result** (SIFTSResult) – Input structures and mapping to use for distance map calculation
- **structures** (*str or dict, optional (default: None)*) – If str: Load structures from directory this string points to. Missing structures will be fetched from web.

  If dict: dictionary with lower-case PDB ids as keys and PDB objects as values. This dictionary has to contain all necessary structures, missing ones will not be fetched. This dictionary can be created using pdb.load_structures.
- **atom_filter** (*str, optional (default: None)*) – Filter coordinates to contain only these atoms. E.g. set to "CA" to compute C_alpha - C_alpha distances instead of minimum atom distance over all atoms in both residues.
- **intersect** (*bool, optional (default: False)*) – If True, intersect indices of the given distance maps. Otherwise, union of indices will be used.
- **output_prefix** (*str, optional (default: None)*) – If given, save individual and final contact maps to files prefixed with this string. The appended file suffixes map to row index in sifts_results.hits
- **model** (*int, optional (default: 0)*) – Index of model in PDB structure that should be used
- **raise_missing** (*bool, optional (default: True)*) – Raise a ResourceError if any of the input structures can not be loaded; otherwise, ignore missing entries.

Returns Computed aggregated distance map across all input structures

Return type *DistanceMap*

Raises

- ValueError – If sifts_result is empty (no structure hits)
- ResourceError – If any structure could not be loaded and raise_missing is True

evcouplings.compare.distances.**multimer_dists**(*sifts_result*, *structures=None*, *atom_filter=None*, *intersect=False*, *output_prefix=None*, *model=0*, *raise_missing=True*)

Compute homomultimer distances (between repeated copies of the same entity) in PDB file. Resulting distance matrix will be symmetric by minimization over upper and lower triangle of matrix, even if the complex structure is not symmetric.

Parameters

- **sifts_result** (SIFTSResult) – Input structures and mapping to use for distance map calculation
- **structures** (*str or dict, optional (default: None)*) – If str: Load structures from directory this string points to. Missing structures will be fetched from web.

If dict: dictionary with lower-case PDB ids as keys and PDB objects as values. This dictionary has to contain all necessary structures, missing ones will not be fetched. This dictionary can be created using pdb.load_structures.

- **atom_filter** (*str, optional (default: None)*) – Filter coordinates to contain only these atoms. E.g. set to "CA" to compute C_alpha - C_alpha distances instead of minimum atom distance over all atoms in both residues.
- **intersect** (*bool, optional (default: False)*) – If True, intersect indices of the given distance maps. Otherwise, union of indices will be used.
- **output_prefix** (*str, optional (default: None)*) – If given, save individual and final contact maps to files prefixed with this string. The appended file suffixes map to row index in sifts_results.hits
- **model** (*int, optional (default: 0)*) – Index of model in PDB structure that should be used
- **raise_missing** (*bool, optional (default: True)*) – Raise a ResourceError if any of the input structures can not be loaded; otherwise, ignore missing entries.

Returns Computed aggregated distance map across all input structures

Return type *DistanceMap*

Raises

- ValueError – If sifts_result is empty (no structure hits)
- ResourceError – If any structure could not be loaded and raise_missing is True

evcouplings.compare.distances.**remap_chains**(*sifts_result*, *output_prefix*, *sequence=None*, *structures=None*, *atom_filter=('N', 'CA', 'C', 'O')*, *model=0*, *chain_name='A'*, *raise_missing=True*)

Remap a set of PDB chains into the numbering scheme (and amino acid sequence) of a target sequence (a.k.a. the poorest homology model possible).

(This function is placed here because of close relationship to intra_dists and reusing functionality for it).

Parameters

- **sifts_result** (SIFTSResult) – Input structures and mapping to use for remapping
- **output_prefix** (*str*) – Save remapped structures to files prefixed with this string
- **sequence** (*dict, optional (default: None)*) – Mapping from sequence position (int or str) to residue. If this parameter is given, residues in the output structures will be renamed to the residues in this mapping.

  Note: if side-chain residues are not taken off using atom_filter, this will e.g. happily label an actual glutamate as an alanine).

- **structures** (*str or dict, optional (default: None)*) –
  - If str: Load structures from directory this string points to. Missing structures will be fetched from web.
  - If dict: dictionary with lower-case PDB ids as keys and PDB objects as values. This dictionary has to contain all necessary structures, missing ones will not be fetched. This dictionary can be created using pdb.load_structures.

- **atom_filter** (*str, optional (default: ("N", "CA", "C", "O"))*) – Filter coordinates to contain only these atoms. If None, will retain all atoms; the default value will only keep backbone atoms.
- **model** (*int, optional (default: 0)*) – Index of model in PDB structure that should be used
- **chain_name** (*str, optional (default: "A")*) – Rename the PDB chain to this when saving the file. This will not affect the file name, only the name of the chain in the PDB object.
- **raise_missing** (*bool, optional (default: True)*) – Raise a ResourceError if any of the input structures can not be loaded; otherwise, ignore missing entries.

Returns **remapped** – Mapping from index of each structure hit in sifts_results.hits to filename of stored remapped structure

Return type dict

evcouplings.compare.distances.**remap_complex_chains**(*sifts_result_i*, *sifts_result_j*, *sequence_i=None*, *sequence_j=None*, *structures=None*, *atom_filter=('N', 'CA', 'C', 'O')*, *output_prefix=None*, *raise_missing=True*, *chain_name_i='A'*, *chain_name_j='B'*, *model=0*)

Remap a pair of PDB chains from the same structure into the numbering scheme (and amino acid sequence) of a target sequence.

Parameters

- **sifts_result_i** (SIFTSResult) – Input structures and mapping to use for remapping
- **sifts_result_j** (SIFTSResult) – Input structures and mapping to use for remapping
- **output_prefix** (*str*) – Save remapped structures to files prefixed with this string
- **sequence_i** (*dict, optional (default: None)*) – Mapping from sequence position (int or str) in the first sequence to residue. If this parameter is given, residues in the output structures will be renamed to the residues in this mapping.

  Note: if side-chain residues are not taken off using atom_filter, this will e.g. happily label an actual glutamate as an alanine).

- **sequence_j** (*dict, optional (default: None)*) – Same as sequence_j for second sequence.
- **structures** (*str or dict, optional (default: None)*) –
  - If str: Load structures from directory this string points to. Missing structures will be fetched from web.
  - If dict: dictionary with lower-case PDB ids as keys and PDB objects as values. This dictionary has to contain all necessary structures, missing ones will not be fetched. This dictionary can be created using pdb.load_structures.

- **atom_filter** (*str, optional (default: ("N", "CA", "C", "O"))*) – Filter coordinates to contain only these atoms. If None, will retain all atoms; the default value will only keep backbone atoms.
- **model** (*int, optional (default: 0)*) – Index of model in PDB structure that should be used
- **raise_missing** (*bool, optional (default: True)*) – Raise a ResourceError if any of the input structures can not be loaded; otherwise, ignore missing entries.
- **chain_name_i** (*str, optional (default: "A")*) – Renames the first chain to this string
- **chain_name_j** (*str, optional (default: "B")*) – Renames the second chain to this string

Returns **remapped** – Mapping from index of each structure hit in sifts_results.hits to filename of stored remapped structure

Return type dict

Raises

- `ValueError` – If sifts_result_i or sifts_result_j is empty (no structure hits)
- `ResourceError` – If any structure could not be loaded and raise_missing is True

2.1.2 evcouplings.compare.ecs module

Compare evolutionary couplings to distances in 3D structures

Authors: Thomas A. Hopf

`evcouplings.compare.ecs.`**add_distances**(*ec_table*, *dist_map*, *target_column='dist'*)

Add pair distances to EC score table

Parameters

- **ec_table** (*pandas.DataFrame*) – List of evolutionary couplings, with pair positions in columns i and j
- **dist_map** (`DistanceMap`) – Distance map that will be used to annotate distances in ec_table
- **target_column** (*str*) – Name of column in which distances will be stored

Returns Couplings table with added distances in target_column. Pairs where no distance information is available will be np.nan

Return type pandas.DataFrame

`evcouplings.compare.ecs.`**add_precision**(*ec_table*, *dist_cutoff=5*, *score='cn'*, *min_sequence_dist=6*, *target_column='precision'*, *dist_column='dist'*)

Compute precision of evolutionary couplings as predictor of 3D structure contacts

Parameters

- **ec_table** (*pandas.DataFrame*) – List of evolutionary couplings
- **dist_cutoff** (*float, optional (default: 5)*) – Upper distance cutoff (in Angstrom) for a pair to be considered a true positive contact

- **score** (*str, optional (default: "cn")*) – Column which contains coupling score. Table will be sorted in descending order by this score.
- **min_sequence_dist** (*int, optional (default: 6)*) – Minimal distance in primary sequence for an EC to be included in precision calculation
- **target_column** (*str, optional (default: "precision")*) – Name of column in which precision will be stored
- **dist_column** (*str, optional (default: "dist")*) – Name of column which contains pair distances

Returns EC table with added precision values as a function of EC rank (returned table will be sorted by score column)

Return type pandas.DataFrame

evcouplings.compare.ecs.**coupling_scores_compared**(*ec_table*, *dist_map*, *dist_map_multimer=None*, *dist_cutoff=5*, *output_file=None*, *score='cn'*, *min_sequence_dist=6*)

Utility function to create "CouplingScores.csv"-style table

Parameters

- **ec_table** (*pandas.DataFrame*) – List of evolutionary couplings
- **dist_map** (DistanceMap) – Distance map that will be used to annotate distances in ec_table
- **dist_map_multimer** (DistanceMap, *optional (default: None)*) – Additional multimer distance map. If given, the distance for any EC pair will be the minimum out of the monomer and multimer distances.
- **dist_cutoff** (*float, optional (default: 5)*) – Upper distance cutoff (in Angstrom) for a pair to be considered a true positive contact
- **output_file** (*str, optional (default: None)*) – Store final table to this file
- **score** (*str, optional (default: "cn")*) – Column which contains coupling score. Table will be sorted in descending order by this score.
- **min_sequence_dist** (*int, optional (default: 6)*) – Minimal distance in primary sequence for an EC to be included in precision calculation

Returns EC table with added distances, and precision if dist_cutoff is given.

Return type pandas.DataFrame

2.1.3 evcouplings.compare.mapping module

Index mapping for PDB structures

Authors: Thomas A. Hopf Charlotta P. Schärfe

evcouplings.compare.mapping.**alignment_index_mapping**(*alignment_file*, *format='stockholm'*, *target_seq=None*)

Create index mapping table between sequence positions based on a sequence alignment.

Parameters

- **alignment_file** (*str*) – Path of alignment file containing sequences for which indices shoul dbe mapped
- **format** (*{"stockholm", "fasta"}*) – Format of alignment file
- **target_seq** (*str, optional (default: None)*) – Identifier of sequence around which the index mapping will be centered. If None, first sequence in alignment will be used.

Returns

Mapping table containing assignment of

1. index in target sequence (i)
2. symbol in target sequence (A_i)

For all other sequences in alignment, the following two columns:

3. index in second sequence (j_<sequence id>)
4. symbol in second sequence (A_j_<sequence_id>)

Return type pandas.DataFrame

evcouplings.compare.mapping.**map_indices**(*seq_i*, *start_i*, *end_i*, *seq_j*, *start_j*, *end_j*, *gaps='-'*, *'.'*)

Compute index mapping between positions in two aligned sequences

Parameters

- **seq_i** (*str*) – First aligned sequence
- **start_i** (*int*) – Index of first position in first sequence
- **end_i** (*int*) – Index of last position in first sequence (used for verification purposes only)
- **seq_j** (*str*) – Second aligned sequence
- **start_j** (*int*) – Index of first position in second sequence
- **end_j** (*int*) – Index of last position in second sequence (used for verification purposes only)

Returns

Mapping table containing assignment of

1. index in first sequence (i)
2. symbol in first sequence (A_i)
3. index in second sequence (j)
4. symbol in second sequence (A_j)

Return type pandas.DataFrame

2.1.4 evcouplings.compare.pdb module

PDB structure handling based on MMTF format

Authors: Thomas A. Hopf

class evcouplings.compare.pdb.**Chain**(*residues*, *coords*)
Bases: object

Container for PDB chain residue and coordinate information

filter_atoms(*atom_name='CA'*)
Filter coordinates of chain, e.g. to compute C_alpha-C_alpha distances

Parameters **atom_name** (*str or list-like, optional (default: "CA")*) – Name(s) of atoms to keep

Returns Chain containing only filtered atoms (and those residues that have such an atom)

Return type *Chain*

filter_positions(*positions*)
Select a subset of positions from the chain

Parameters **positions** (*list-like*) – Set of residues that will be kept

Returns Chain containing only the selected residues

Return type *Chain*

remap(*mapping*, *source_id='seqres_id'*)
Remap chain into different numbering scheme (e.g. from seqres to uniprot numbering)

Parameters

- **mapping** (*dict*) – Mapping of residue identifiers from source_id (current main index of PDB chain) to new identifiers.

  mapping may either be:

  1. dict(str -> str) to map individual residue IDs. Keys and values of dictionary will be typecast to string before the mapping, so it is possible to pass in integer values too (if the source or target IDs are numbers)
  2. dict((int, int) -> (int, int)) to map ranges of numbers to ranges of numbers. This should typically be only used with RESSEQ or UniProt numbering. End index or range is *inclusive* Note that residue IDs in the end will still be handled as strings when mapping.

- **source_id** (*{"seqres_id", "coord_id", "id"}, optional (default: "seqres_id")*) – Residue identifier in chain to map *from* (will be used as key to access mapping)

Returns Chain with remapped numbering ("id" column in residues DataFrame)

Return type *Chain*

to_file(*fileobj*, *chain_id='A'*, *end=True*, *first_atom_id=1*)
Write chain to a file in PDB format (mmCIF not yet supported).

Note that PDB files written this function may not be 100% compliant with the PDB format standards, in particular:

- some HETATM records may turn into ATOM records when starting from an mmtf file, if the record has a one-letter code (such as MSE / M).
- code does not print TER record at the end of a peptide chain

Parameters

- **fileobj** (*file-like object*) – Write to this file handle

- **chain_id** (*str, optional (default: "A")*) – Assign this chain name in file (allows to redefine chain name from whatever chain was originally)
- **end** (*bool, optional (default: True)*) – Print "END" record after chain (signals end of PDB file)
- **first_atom_id** (*int, optional (default: 1)*) – Renumber atoms to start with this index (set to None to keep default indices)

Raises ValueError – If atom or residue numbers are too wide and cannot be written to old fixed-column PDB file format

to_seqres()
Return copy of chain with main index set to SEQRES numbering. Residues that do not have a SEQRES id will be dropped.

Returns Chain with seqres IDs as main index

Return type *Chain*

class evcouplings.compare.pdb.**ClassicPDB**(*structure*)
Bases: object

Class to handle "classic" PDB and mmCIF formats (for new mmtf format see PDB class above). Wraps around Biopython PDB functionality to provide a consistent interface.

Unlike the PDB class (based on mmtf), this object will not be able to extract SEQRES indices corresponding to ATOM-record residue indices.

classmethod from_file(*filename*, *file_format='pdb'*)
Initialize structure from PDB/mmCIF file

Parameters

- **filename** (*str*) – Path of file
- **file_format** (*{"pdb", "cif"}, optional (default: "pdb")*) – Format of structure (old PDB format or mmCIF)

Returns Initialized PDB structure

Return type *ClassicPDB*

classmethod from_id(*pdb_id*)
Initialize structure by PDB ID (fetches structure from RCSB servers)

Parameters **pdb_id** (*str*) – PDB identifier (e.g. 1hzx)

Returns initialized PDB structure

Return type *PDB*

get_chain(*chain*, *model=0*)
Extract residue information and atom coordinates for a given chain in PDB structure

Parameters

- **chain** (*str*) – Name of chain to be extracted (e.g. "A")
- **model** (*int, optional (default: 0)*) – Index of model to be extracted

Returns Chain object containing DataFrames listing residues and atom coordinates

Return type *Chain*

class evcouplings.compare.pdb.**PDB**(*mmtf*)

Bases: object

Wrapper around PDB MMTF decoder object to access residue and coordinate information

classmethod from_file(*filename*)

Initialize structure from MMTF file

Parameters **filename** (*str*) – Path of MMTF file

Returns initialized PDB structure

Return type *PDB*

classmethod from_id(*pdb_id*)

Initialize structure by PDB ID (fetches structure from RCSB servers)

Parameters **pdb_id** (*str*) – PDB identifier (e.g. 1hzx)

Returns initialized PDB structure

Return type *PDB*

get_chain(*chain*, *model=0*)

Extract residue information and atom coordinates for a given chain in PDB structure

Parameters

- **chain** (*str*) – Name of chain to be extracted (e.g. "A")
- **model** (*int, optional (default: 0)*) – Index of model to be extracted

Returns Chain object containing DataFrames listing residues and atom coordinates

Return type *Chain*

evcouplings.compare.pdb.**load_structures**(*pdb_ids*, *structure_dir=None*, *raise_missing=True*)

Load PDB structures from files / web

Parameters

- **pdb_ids** (*Iterable*) – List / iterable containing PDB identifiers to be loaded.
- **structure_dir** (*str, optional (default: None)*) – Path to directory with structures. Structures filenames must be in the format 5p21.mmtf. If a file can not be found, will try to fetch from web instead.
- **raise_missing** (*bool, optional (default: True)*) – Raise a ResourceError exception if any of the PDB IDs cannot be loaded. If False, missing entries will be ignored.

Returns **structures** – Dictionary containing loaded structures. Keys (PDB identifiers) will be lowercase.

Return type dict(str -> PDB)

Raises ResourceError – Raised if raise_missing is True and any of the given PDB IDs cannot be loaded.

2.1.5 evcouplings.compare.protocol module

2.1.6 evcouplings.compare.sifts module

Uniprot to PDB structure identification and index mapping using the SIFTS database (https://www.ebi.ac.uk/pdbe/docs/sifts/)

This functionality is centered around the pdb_chain_uniprot.csv table available from SIFTS. (ftp://ftp.ebi.ac.uk/pub/databases/msd/sifts/flatfiles/csv/pdb_chain_uniprot.csv.gz)

Authors: Thomas A. Hopf Anna G. Green (find_homologs) Chan Kang (find_homologs)

class evcouplings.compare.sifts.**SIFTS**(*sifts_table_file*, *sequence_file=None*)
Bases: object

Provide Uniprot to PDB mapping data and functions starting from SIFTS mapping table.

by_alignment(*min_overlap=20*, *reduce_chains=False*, ***kwargs*)
Find structures by sequence alignment between query sequence and sequences in PDB.

Parameters

- **min_overlap** (*int, optional (default: 20)*) – Require at least this many aligned positions with the target structure
- **reduce_chains** (*bool, optional (Default: True)*) – If true, keep only first chain per PDB ID (i.e. remove redundant occurrences of same protein in PDB structures). Should be set to False to identify homomultimeric contacts.
- ****kwargs** – Defines the behaviour of find_homologs() function used to find homologs by sequence alignment: - which alignment method is used

  (pdb_alignment_method: {"jackhmmer", "hmmsearch"}, default: "jackhmmer"),

  – parameters passed into the protocol for the selected alignment method (evcouplings.align.jackhmmer_search or evcouplings.align.hmmbuild_and_search).

  Default parameters are set in the HMMER_CONFIG string in this module, other parameters will need to be overriden; these minimally are: - for pdb_alignment_method == "jackhmmer":

  * sequence_id : str, identifier of target sequence
  * jackhmmer : str, path to jackhmmer binary if not on path
  * for pdb_alignment_method == "hmmsearch": - sequence_id : str, identifier of target sequence - raw_focus_alignment_file : str, path to input alignment file - hmmbuild : str, path to hmmbuild binary if not on path - hmmsearch : str, path to search binary if not on path

  – additionally, if "prefix" is given, individual mappings will be saved to files suffixed by the respective key in mapping table.

Returns Record of hits and mappings found for this query sequence by alignment. See by_pdb_id() for detailed explanation of fields.

Return type *SIFTSResult*

by_pdb_id(*pdb_id*, *pdb_chain=None*, *uniprot_id=None*)
Find structures and mapping by PDB id and chain name

Parameters

- **pdb_id** (*str*) – 4-letter PDB identifier
- **pdb_chain** (*str, optional (default: None)*) – PDB chain name (if not given, all chains for PDB entry will be returned)
- **uniprot_id** (*str, optional (default: None)*) – Filter to keep only this Uniprot accession number or identifier (necessary for chimeras, or multi-chain complexes with different proteins)

Returns Identified hits plus index mappings to Uniprot

Return type *SIFTSResult*

Raises ValueError – If selected segments in PDB file do not unambigously map to one Uniprot entry

by_uniprot_id(*uniprot_id*, *reduce_chains=False*)

Find structures and mapping by Uniprot access number.

Parameters

- **uniprot_ac** (*str*) – Find PDB structures for this Uniprot accession number. If sequence_file was given while creating the SIFTS object, Uniprot identifiers can also be used.
- **reduce_chains** (*bool, optional (Default: True)*) – If true, keep only first chain per PDB ID (i.e. remove redundant occurrences of same protein in PDB structures). Should be set to False to identify homomultimeric contacts.

Returns Record of hits and mappings found for this Uniprot protein. See by_pdb_id() for detailed explanation of fields.

Return type *SIFTSResult*

create_sequence_file(*output_file*, *chunk_size=1000*, *max_retries=100*)

Create FASTA sequence file containing all UniProt sequences of proteins in SIFTS. This file is required for homology-based structure identification and index remapping. This function will also automatically associate the sequence file with the SIFTS object.

Parameters

- **output_file** (*str*) – Path at which to store sequence file
- **chunk_size** (*int, optional (default: 1000)*) – Retrieve sequences from UniProt in chunks of this size (too large chunks cause the mapping service to stall)
- **max_retries** (*int, optional (default: 100)*) – Allow this many retries when fetching sequences from UniProt ID mapping service, which unfortunately often suffers from connection failures.

class evcouplings.compare.sifts.**SIFTSResult**(*hits*, *mapping*)

Bases: object

Store results of SIFTS structure/mapping identification.

(Full class defined for easify modification of fields)

evcouplings.compare.sifts.**fetch_uniprot_mapping**(*ids*, *from_='ACC'*, *to='ACC'*, *format='fasta'*)

Fetch data from UniProt ID mapping service (e.g. download set of sequences)

Parameters

- **ids** (*list(str)*) – List of UniProt identifiers for which to retrieve mapping

- **from** (*str, optional (default: "ACC")*) – Source identifier (i.e. contained in "ids" list)
- **to** (*str, optional (default: "ACC")*) – Target identifier (to which source should be mapped)
- **format** (*str, optional (default: "fasta")*) – Output format to request from Uniprot server

Returns Response from UniProt server

Return type str

evcouplings.compare.sifts.**find_homologs**(*pdb_alignment_method='jackhmmer'*, ***kwargs*)

Identify homologs using jackhmmer or hmmbuild/hmmsearch

Parameters

- **pdb_alignment_method** (*{"jackhmmer", "hmmsearch"},*) – optional (default: "jackhmmer") Sequence alignment method used for searching the PDB
- ****kwargs** – Passed into jackhmmer / hmmbuild_and_search protocol (see documentation for available options)

Returns

- **ali** (*evcouplings.align.Alignment*) – Alignment of homologs of query sequence in sequence database
- **hits** (*pandas.DataFrame*) – Tabular representation of hits

2.2 evcouplings.complex package

2.2.1 evcouplings.complex.protocol module

Protocols for matching putatively interacting sequences in protein complexes to create a concatenated sequence alignment

Authors: Anna G. Green Thomas A. Hopf

evcouplings.complex.protocol.**best_hit**(***kwargs*)

Protocol:

Concatenate alignments based on the best hit to the focus sequence in each species

Parameters **kwargs arguments** (*Mandatory*) – See list below in code where calling check_required

Returns

outcfg – Output configuration of the pipeline, including the following fields:

alignment_file raw_alignment_file focus_mode focus_sequence segments frequencies_file identities_file num_sequences num_sites raw_focus_alignment_file statistics_file

Return type dict

evcouplings.complex.protocol.**describe_concatenation**(*annotation_file_1*, *annotation_file_2*, *genome_location_filename_1*, *genome_location_filename_2*, *outfile*)

Describes properties of concatenated alignment.

Writes a csv with the following columns

num_seqs_1 : number of sequences in the first monomer alignment num_seqs_2 : number of sequences in the second monomer alignment num_nonred_species_1 : number of unique species annotations in the

first monomer alignment

num_nonred_species_2 [number of unique species annotations in the] second monomer alignment

num_species_overlap: number of unique species found in both alignments median_num_per_species_1 : median number of paralogs per species in the

first monomer alignmment

median_num_per_species_2 [median number of paralogs per species in] the second monomer alignment

num_with_embl_cds_1 [number of IDs for which we found an EMBL CDS in the] first monomer alignment (relevant to distance concatention only)

num_with_embl_cds_2 [number of IDs for which we found an EMBL CDS in the] first monomer alignment (relevant to distance concatention only)

Parameters

- **annotation_file_1** (*str*) – Path to annotation.csv file for first monomer alignment
- **annotation_file_2** (*str*) – Path to annotation.csv file for second monomer alignment
- **genome_location_filename_1** (*str*) – Path to genome location mapping file for first alignment
- **genome_location_filename_2** (*str*) – Path to genome location mapping file for second alignment
- **outfile** (*str*) – Path to output file

evcouplings.complex.protocol.**genome_distance**(***kwargs*)

Protocol:

Concatenate alignments based on genomic distance

Parameters **kwargs arguments** (*Mandatory*) – See list below in code where calling check_required

Returns

outcfg – Output configuration of the pipeline, including the following fields:

- alignment_file
- raw_alignment_file
- focus_mode
- focus_sequence

- segments
- frequencies_file
- identities_file
- num_sequences
- num_sites
- raw_focus_alignment_file
- statistics_file

Return type dict

evcouplings.complex.protocol.**modify_complex_segments**(*outcfg*, ***kwargs*)

Modifies the output configuration so that the segments are correct for a concatenated alignment

Parameters **outcfg** (*dict*) – The output configuration

Returns **outcfg** – The output configuration, with a new field called "segments"

Return type dict

evcouplings.complex.protocol.**run**(***kwargs*)

Run alignment concatenation protocol

Parameters **kwargs arguments** (*Mandatory*) – protocol: concatenation protocol to run prefix: Output prefix for all generated files

Returns

outcfg – Output configuration of concatenation stage Dictionary with results in following fields: (in brackets: not mandatory)

alignment_file raw_alignment_file focus_mode focus_sequence segments frequencies_file identities_file num_sequences num_sites raw_focus_alignment_file statistics_file

Return type dict

2.3 evcouplings.couplings package

2.3.1 evcouplings.couplings.mapping module

Mapping indices for complexes / multi-domain sequences to internal model numbering.

Authors: Thomas A. Hopf Anna G. Green (MultiSegmentCouplingsModel)

class evcouplings.couplings.mapping.**MultiSegmentCouplingsModel**(*filename*, **segments*, *precision='float32'*, *file_format='plmc_v2'*, ***kwargs*)

Bases: *evcouplings.couplings.model.CouplingsModel*

Complex specific Couplings Model that handles segments and provides the option to convert model into inter-segment only.

to_inter_segment_model()

Convert model to inter-segment only parameters, ie the J_ijs that correspond to inter-protein or inter-domain residue pairs. All other parameters are set to 0.

Returns Copy of object turned into inter-only Epistatic model

Return type *CouplingsModel*

class evcouplings.couplings.mapping.**Segment**(*segment_type*, *sequence_id*, *region_start*, *region_end*, *positions=None*, *segment_id='A'*)

Bases: object

Represents a continuous stretch of sequence in a sequence alignment to infer evolutionary couplings (e.g. multiple domains, or monomers in a concatenated complex alignment)

default_chain_name()

Retrieve default PDB chain identifier the segment will be mapped to in 3D structures (by convention, segments in the pipeline are named A_1, A_2, ..., B_1, B_2, ...; the default chain identifier is anything before the underscore).

Returns **chain** – Default PDB chain identifier the segment maps to

Return type str

classmethod from_list(*segment*)

Create a segment object from list representation (e.g. from config).

Parameters **segment** (*list*) – List representation of segment, with the following items: segment_id (str), segment_type (str), sequence_id (str), region_start (int), region_end (int), positions (list(int))

Returns New Segment instance from list

Return type *Segment*

to_list()

Represent segment as list (for storing in configs)

Returns List representation of segment, with the following items: segment_id (str), segment_type (str), sequence_id (str), region_start (int), region_end (int), positions (list(int))

Return type list

class evcouplings.couplings.mapping.**SegmentIndexMapper**(*focus_mode*, *first_index*, **segments*)

Bases: object

Map indices of one or more sequence segments into CouplingsModel internal numbering space. Can also be used to (trivially) remap indices for a single sequence.

patch_model(*model*, *inplace=True*)

Change numbering of CouplingModel object so that it uses segment-based numbering

Parameters

- **model** (CouplingsModel) – Model that will be updated to segment- based numbering
- **inplace** (*bool, optional (default: True)*) – If True, change passed model; otherwise returnnew object

Returns Model with updated numbering (if inplace is False, this will point to original model)

Return type *CouplingsModel*

Raises ValueError – If segment mapping does not match internal model numbering

to_model(*x*)

Map target index to model index

Parameters **x** (*(str, int), or list of (str, int)*) – Indices in target indexing (segment_id, index_in_segment)

Returns Monomer indices mapped into couplings object numbering

Return type int, or list of int

to_target(*x*)
Map model index to target index

Parameters **x** (*int, or list of ints*) – Indices in model numbering

Returns Indices mapped into target numbering. Tuples are (segment_id, index_in_segment)

Return type (str, int), or list of (str, int)

evcouplings.couplings.mapping.**segment_map_ecs**(*ecs*, *mapper*)
Map EC dataframe in model numbering into segment numbering

Parameters **ecs** (*pandas.DataFrame*) – EC table (with columns i and j)

Returns Mapped EC table (with columns i and j mapped, and additional columns segment_i and segment_j)

Return type pandas.DataFrame

2.3.2 evcouplings.couplings.mean_field module

Inference of evolutionary couplings from sequence alignments using mean field approximation.

Authors: Sophia F. Mersmann

class evcouplings.couplings.mean_field.**MeanFieldCouplingsModel**(*alignment*, *index_list*, *regularized_f_i*, *regularized_f_ij*, *h_i*, *J_ij*, *theta*, *pseudo_count*)
Bases: *evcouplings.couplings.model.CouplingsModel*

Mean field DCA specific model class that stores the parameters inferred using mean field approximation and calculates mutual and direct information as well as fn and cn scores.

di_scores
L x L numpy matrix with DI (direct information) scores

regularize_f_i()
Returns single-site frequencies regularized by a pseudo-count of symbols in alignment.

This method sets the attribute self.regularized_f_i and returns a reference to it.

Returns Matrix of size L x num_symbols containing relative column frequencies of all symbols regularized by a pseudo-count.

Return type np.array

regularize_f_ij()
Returns pairwise frequencies regularized by a pseudo-count of symbols in alignment.

This method sets the attribute self.regularized_f_ij and returns a reference to it.

Returns Matrix of size L x L x num_symbols x num_symbols containing relative pairwise frequencies of all symbols regularized by a pseudo-count.

Return type np.array

tilde_fields(*i*, *j*)
Compute h_tilde fields of the two-site model.

Parameters

- **i** (*int*) – First position.
- **j** (*int*) – Second position.

Returns h_tilde fields of position i and j - both arrays of size 1 x num_symbols

Return type np.array, np.array

to_file(*out_file*, *precision='float32'*, *file_format='plmc_v2'*)
Writes the model to binary file.

This method overrides its respective parent method in CouplingsModel.

Parameters

- **out_file** (*str*) – A string specifying the path to a file
- **precision** (*{"float16", "float32", "float64"}, optional (default: "float32")*) – Numerical NumPy data type specifying the precision used to write numerical values to file
- **file_format** (*{"plmc_v2"}, optional (default: "plmc_v2")*) – For now, a mean-field model can only be written to a file in plmc_v2 format. Writing to a plmc_v1 file is not permitted since there is no functionality provided to read a mean-field model in plmc_v1 format.

to_independent_model()
Compute a single-site model.

This method overrides its respective parent method in CouplingsModel.

Returns Copy of object turned into independent model

Return type *MeanFieldCouplingsModel*

to_raw_ec_file(*couplings_file*)
Write mutual and direct information to the EC file.

Parameters **couplings_file** (*str*) – Output path for file with evolutionary couplings.

transform_from_plmc_model()
Adaptions that allow to read a mean-field couplings model from file where __read_plmc_v2 in CouplingsModel does the heavy lifting.

This includes: - Manage unused plmc-specific fields as well as the pseudo count field - Modify pair frequencies - Regularize column and pair frequencies (i.e. fill the fields regularized_f_i and regularized_f_ij)

class evcouplings.couplings.mean_field.**MeanFieldDCA**(*alignment*)
Bases: object

Class that provides the functionality to infer evolutionary couplings from a given sequence alignment using mean field approximation.

Important: The input alignment should be an a2m alignment with lower / upper columns and the target sequence as the first record.

_raw_alignment
Alignment – The input alignment. This should be an a2m alignment with lower / upper columns and the target sequence as first record.

index_list
np.array – List of UniProt numbers of the target sequence (only upper case characters).

alignment
Alignment – A processed version of the given alignment (_raw_alignment) that is then used to infer evolutionary couplings using DCA.

N
int – The number of sequences (of the processed alignment).

L
int – The width of the alignment (again, this refers to the processed alignment).

num_symbols
int – The number of symbols of the alphabet used.

covariance_matrix
np.array – Matrix of size (L * (num_symbols-1)) x (L * (num_symbols-1)) containing the co-variation of each character pair in any positions.

covariance_matrix_inv
np.array – Inverse of covariance_matrix.

compute_covariance_matrix()
Compute the covariance matrix.

This method sets the attribute self.covariance_matrix and returns a reference to it.

Returns Reference to attribute self.convariance_matrix

Return type np.array

fields()
Compute fields.

Returns Matrix of size L x num_symbols containing single-site fields.

Return type np.array

fit(*theta=0.8*, *pseudo_count=0.5*)
Run mean field direct couplings analysis.

Parameters

- **theta** (*float, optional (default: 0.8)*) – Sequences with pairwise identity >= theta will be clustered and their sequence weights downweighted as 1 / num_cluster_members.
- **pseudo_count** (*float, optional (default: 0.5)*) – Applied to frequency counts to regularize in the case of insufficient data availability.

Returns Model object that holds the inferred fields (h_i) and couplings (J_ij).

Return type *MeanFieldCouplingsModel*

regularize_frequencies(*pseudo_count=0.5*)
Returns single-site frequencies regularized by a pseudo-count of symbols in alignment.

This method sets the attribute self.regularized_frequencies and returns a reference to it.

Parameters **pseudo_count** (*float, optional (default: 0.5)*) – The value to be added as pseudo-count.

Returns Matrix of size L x num_symbols containing relative column frequencies of all symbols regularized by a pseudo-count.

Return type np.array

regularize_pair_frequencies(*pseudo_count=0.5*)
Add pseudo-count to pairwise frequencies to regularize in the case of insufficient data availability.

This method sets the attribute self.regularized_pair_frequencies and returns a reference to it.

Parameters **pseudo_count** (*float, optional (default: 0.5)*) – The value to be added as pseudo-count.

Returns Matrix of size L x L x num_symbols x num_symbols containing relative pairwise frequencies of all symbols regularized by a pseudo-count.

Return type np.array

reshape_invC_to_4d()
"Un-flatten" inverse of the covariance matrix to allow easy access to couplings using position and symbol indices.

Returns Matrix of size L x L x num_symbols x num_symbols.

Return type np.array

evcouplings.couplings.mean_field.**regularize_frequencies**(*f_i*, *pseudo_count=0.5*)
Returns/calculates single-site frequencies regularized by a pseudo-count of symbols in alignment.

Parameters

- **f_i** (*np.array*) – Matrix of size L x num_symbols containing column frequencies.
- **pseudo_count** (*float, optional (default: 0.5)*) – The value to be added as pseudo-count.

Returns Matrix of size L x num_symbols containing relative column frequencies of all symbols regularized by a pseudo-count.

Return type np.array

evcouplings.couplings.mean_field.**regularize_pair_frequencies**(*f_ij*, *pseudo_count=0.5*)
Add pseudo-count to pairwise frequencies to regularize in the case of insufficient data availability.

Parameters

- **f_ij** (*np.array*) – Matrix of size L x L x num_symbols x num_symbols containing pair frequencies.
- **pseudo_count** (*float, optional (default: 0.5)*) – The value to be added as pseudo-count.

Returns Matrix of size L x L x num_symbols x num_symbols containing relative pairwise frequencies of all symbols regularized by a pseudo-count.

Return type np.array

2.3.3 evcouplings.couplings.model module

Class to store parameters of undirected graphical model of sequences and perform calculations using the model (statistical energies, coupling scores).

Authors: Thomas A. Hopf

class evcouplings.couplings.model.**CouplingsModel**(*filename*, *precision='float32'*, *file_format='plmc_v2'*, ***kwargs*)

Bases: object

Class to store parameters of pairwise undirected graphical model of sequences and compute evolutionary couplings, sequence statistical energies, etc.

Jij(*i=None*, *j=None*, *A_i=None*, *A_j=None*)

Quick access to J_ij matrix with automatic index mapping. See __4d_access for explanation of parameters.

classmethod apc(*matrix*)

Apply average product correction (Dunn et al., Bioinformatics, 2008) to matrix

Parameters **matrix** (*np.array*) – Symmetric L x L matrix which should be corrected by APC

Returns Symmetric L x L matrix with APC correction applied

Return type np.array

cn(*i=None*, *j=None*)

Quick access to cn_scores matrix with automatic index mapping. See __2d_access_score_matrix for explanation of parameters.

cn_scores

L x L numpy matrix with CN (corrected norm) scores

convert_sequences(*sequences*)

Converts sequences in string format into internal symbol representation according to alphabet of model

Parameters **sequences** (*list of str*) – List of sequences (must have same length and correspond to model states)

Returns Matrix of size len(sequences) x L of sequences converted to integer symbols

Return type np.array

delta_hamiltonian(*substitutions*, *verify_mutants=True*)

Calculate difference in statistical energy relative to self.target_seq by changing sequence according to list of substitutions

Parameters

- **substitutions** (*list of tuple(pos, subs_from, subs_to)*) – Substitutions to be applied to target sequence
- **verify_mutants** (*bool, optional*) – Test if subs_from is consistent with self.target_seq

Returns Vector of length 3 with 1) total delta Hamiltonian, 2) delta J_ij, 3) delta h_i

Return type np.array

dmm(*i=None*, *j=None*, *A_i=None*, *A_j=None*)

Access delta_Hamiltonian matrix of double mutants of target sequence

Parameters

- **i** (*Iterable(int) or int*) – Position(s) of first substitution(s)
- **j** (*Iterable(int) or int*) – Position(s) of second substitution(s)
- **A_i** (*Iterable(char) or char*) – Substitution(s) to first position
- **A_j** (*Iterable(char) or char*) – Substitution(s) to second position

Returns 4D matrix containing energy differences for slices along both axes of double mutation matrix (axes 1/2: position, axis 3/4: substitutions).

Return type np.array(float)

double_mut_mat
Hamiltonian difference for all possible double mutant variants

L x L x num_symbol x num_symbol matrix containing delta Hamiltonians for all possible double mutants of target sequence

ecs
DataFrame with evolutionary couplings, sorted by CN score (all scores: CN, FN, MI)

fi(*i=None*, *A_i=None*)
Quick access to f_i matrix with automatic index mapping. See __2d_access for explanation of parameters.

fij(*i=None*, *j=None*, *A_i=None*, *A_j=None*)
Quick access to f_ij matrix with automatic index mapping. See __4d_access for explanation of parameters.

fn(*i=None*, *j=None*)
Quick access to fn_scores matrix with automatic index mapping. See __2d_access_score_matrix for explanation of parameters.

fn_scores
L x L numpy matrix with FN (Frobenius norm) scores

hamiltonians(*sequences*)
Calculates the Hamiltonians of the global probability distribution P(A_1, ..., A_L) for the given sequences A_1,...,A_L from J_ij and h_i parameters

Parameters **sequences** (*list of str*) – List of sequences for which Hamiltonian will be computed, or converted np.array obtained using convert_sequences method

Returns Float matrix of size len(sequences) x 3, where each row corresponds to the 1) total Hamiltonian of sequence and the 2) J_ij and 3) h_i sub-sums

Return type np.array

hi(*i=None*, *A_i=None*)
Quick access to h_i matrix with automatic index mapping. See __2d_access for explanation of parameters.

index_list
Target/Focus sequence of model used for delta_hamiltonian calculations (including single and double mutation matrices)

itu(*i=None*)
Legacy method for backwards compatibility. See self.sn for explanation.

mi_apc(*i=None*, *j=None*)
Quick access to mi_scores_apc matrix with automatic index mapping. See __2d_access_score_matrix for explanation of parameters.

mi_raw(*i=None*, *j=None*)
Quick access to mi_scores_raw matrix with automatic index mapping. See __2d_access_score_matrix for explanation of parameters.

mi_scores_apc
L x L numpy matrix with MI (mutual information) scores with APC correction

mi_scores_raw
L x L numpy matrix with MI (mutual information) scores without APC correction

mn(*i=None*)
Map model numbering to internal numbering

Parameters **i** (*Iterable(int) or int*) – Position(s) to be mapped from model numbering space into internal numbering space

Returns Remapped position(s)

Return type Iterable(int) or int

mui(*i=None*)
Legacy method for backwards compatibility. See self.mn for explanation.

seq(*i=None*)
Access target sequence of model

Parameters **i** (*Iterable(int) or int*) – Position(s) for which symbol should be retrieved

Returns Sequence symbols

Return type Iterable(char) or char

single_mut_mat
Hamiltonian difference for all possible single-site variants

L x num_symbol matrix (np.array) containing delta Hamiltonians for all possible single mutants of target sequence.

single_mut_mat_full
Hamiltonian difference for all possible single-site variants

L x num_symbol x 3 matrix (np.array) containing delta Hamiltonians for all possible single mutants of target sequence. Third dimension: 1) full Hamiltonian, 2) J_ij, 3) h_i

smm(*i=None*, *A_i=None*)
Access delta_Hamiltonian matrix of single mutants of target sequence

Parameters

- **i** (*Iterable(int) or int*) – Position(s) for which energy difference should be retrieved
- **A_i** (*Iterable(char) or char*) – Substitutions for which energy difference should be retrieved

Returns 2D matrix containing energy differences for slices along both axes of single mutation matrix (first axis: position, second axis: substitution).

Return type np.array(float)

sn(*i=None*)
Map internal numbering to sequence numbering

Parameters **i** (*Iterable(int) or int*) – Position(s) to be mapped from internal numbering space into sequence numbering space.

Returns Remapped position(s)

Return type Iterable(int) or int

target_seq
Target/Focus sequence of model used for delta_hamiltonian calculations (including single and double mutation matrices)

to_file(*out_file*, *precision='float32'*, *file_format='plmc_v2'*)
Writes the potentially modified model again to binary file

Parameters

- **out_file** (*str*) – A string specifying the path to a file
- **precision** (*{"float16", "float32", "float64"}, optional (default: "float32")*) – Numerical NumPy data type specifying the precision used to write numerical values to file
- **file_format** (*{"plmc_v1", "plmc_v2"}, optional (default: "plmc_v2")*) – Available file formats

to_independent_model()
Estimate parameters of a single-site model using Gaussian prior/L2 regularization.

Returns Copy of object turned into independent model

Return type *CouplingsModel*

2.3.4 evcouplings.couplings.pairs module

Functions for handling evolutionary couplings data.

Todo:

1. clean up
2. add Pompom score
3. add mapping tools (multidomain, complexes)
4. ECs to matrix
5. APC on subsets of positions (e.g. for complexes)

Authors: Thomas A. Hopf Agnes Toth-Petroczy (original mixture model code) John Ingraham (skew normal mixture model) Anna G. Green (EVComplex Score code)

class evcouplings.couplings.pairs.**EVComplexScoreModel**(*x*)
Bases: object

Assign to each EC score a (unnormalized) EVcomplex score as described in Hopf, Schärfe et al. (2014).

TODO: this implementation currently does not take into account score normalization for the number of sequences and length of the model

probability(*x*, *plot=False*)
Calculates evcomplex score as cn_score / min_cn_score. TODO: plotting functionality not yet implemented

Parameters

- **x** (*np.array (or list-like)*) – List of scores
- **plot** (*bool, optional (default: False)*) – Plot score distribution

Returns **probability** – EVcomplex score

Return type np.array(float)

class evcouplings.couplings.pairs.**LegacyScoreMixtureModel**(*x*, *clamp_mu=False*, *max_fun=10000*, *max_iter=1000*)

Bases: object

Assign to each EC score the probability of being in the lognormal tail of a normal-lognormal mixture model.

Note: this is the original version of the score mixture model with a normal distribution noise component, this has been superseded by a model using a skew normal distribution

probability(*x*, *plot=False*)

Calculate posterior probability of EC pair to be located in positive (lognormal) tail of the distribution.

Parameters

- **x** (*np.array (or list-like)*) – List of scores
- **plot** (*bool, optional (default: False)*) – Plot score distribution and probabilities

Returns **posterior** – Posterior probability of being in signal component of mixture model

Return type np.array(float)

class evcouplings.couplings.pairs.**ScoreMixtureModel**(*x*)

Bases: object

Assign to each EC score the probability of being in the lognormal tail of a skew normal-lognormal mixture model.

classmethod lognorm_pdf(*x*, *logmu*, *logsig*)

Probability density of lognormal distribution (signal component)

Parameters

- **x** (*np.array (float)*) – Data for which probability density should be calculated
- **logmu** (*float*) – Location of lognormal distribution (signal component)
- **logsig** (*float*) – Scale parameter of lognormal distribution (signal component)

Returns **density** – Probability density for input array x

Return type np.array(float)

classmethod mixture_pdf(*x*, *p*, *scale*, *skew*, *logmu*, *logsig*)

Compute mixture probability

Parameters

- **x** (*np.array (float)*) – Data for which probability density should be calculated
- **p** (*float*) – Mixing fraction between components for noise component (signal component will be 1-p)
- **scale** (*float*) – Scale parameter of skew normal distribution (noise component)
- **skew** (*float*) – Skew parameter of skew normal distribution (noise component)
- **logmu** (*float*) – Location of lognormal distribution (signal component)
- **logsig** (*float*) – Scale parameter of lognormal distribution (signal component)

Returns **density** – Probability density for input array x

Return type np.array(float)

classmethod posterior_signal(*x*, *p*, *scale*, *skew*, *logmu*, *logsig*)
Compute posterior probability of being in signal component

Parameters

- **x** (*np.array(float)*) – Data for which probability density should be calculated
- **p** (*float*) – Mixing fraction between components for noise component (signal component will be 1-p)
- **scale** (*float*) – Scale parameter of skew normal distribution (noise component)
- **skew** (*float*) – Skew parameter of skew normal distribution (noise component)
- **logmu** (*float*) – Location of lognormal distribution (signal component)
- **logsig** (*float*) – Scale parameter of lognormal distribution (signal component)

Returns **posterior** – Posterior probability of being in signal component for input array x

Return type np.array(float)

probability(*x*, *plot=False*)
Calculate posterior probability of EC pair to be located in positive (lognormal) tail of the distribution.

Parameters **x** (*np.array (or list-like)*) – List of scores

Returns **posterior** – Posterior probability of being in signal component of mixture model

Return type np.array(float)

classmethod skewnorm_constraint(*scale*, *skew*)
Given scale and skew, returns location parameter to yield mean zero

Parameters

- **scale** (*float*) – Scale parameter of skew normal distribution
- **skew** (*float*) – Skew parameter of skew normal distribution

Returns **location** – Location parameter of skew normal distribution s.t. mean of distribution is equal to 0

Return type float

classmethod skewnorm_pdf(*x*, *location*, *scale*, *skew*)
Probability density of skew normal distribution (noise component)

Parameters

- **x** (*np.array(float)*) – Data for which probability density should be calculated
- **location** (*float*) – Location parameter of skew normal distribution
- **scale** (*float*) – Scale parameter of skew normal distribution
- **skew** (*float*) – Skew parameter of skew normal distribution

Returns **density** – Probability density for input array x

Return type np.array(float)

evcouplings.couplings.pairs.**add_mixture_probability**(*ecs*, *model='skewnormal'*, *score='cn'*, *clamp_mu=False*, *plot=False*)
Add lognormal mixture model probability to EC table.

Parameters

- **ecs** (*pd.DataFrame*) – EC table with scores
- **model** (*{"skewnormal", "normal"}, optional (default: skewnormal)*) – Use model with skew-normal or normal distribution for the noise component of mixture model
- **score** (*str, optional (default: "cn")*) – Score on which mixture model will be based
- **clamp_mu** (*bool, optional (default: False)*) – Fix mean of Gaussian component to 0 instead of fitting it based on data
- **plot** (*bool, optional (default: False)*) – Plot score distribution and probabilities

Returns **ec_prob** – EC table with additional column "probability" that for each EC contains the posterior probability of belonging to the lognormal tail of the distribution.

Return type pd.DataFrame

evcouplings.couplings.pairs.**enrichment**(*ecs*, *num_pairs=1.0*, *score='cn'*, *min_seqdist=6*)

Calculate EC "enrichment" as first described in Hopf et al., Cell, 2012.

Todo: Make this handle segments if they are in EC table

Parameters

- **ecs** (*pd.DataFrame*) – Dataframe containing couplings
- **num_pairs** (*int or float, optional (default: 1.0)*) – Number of ECs to use for enrichment calculation. - If float, will be interpreted as fraction of the length of the sequence (e.g. 1.0*L) - If int, will be interpreted as absolute number of pairs
- **score** (*str, optional (default: cn)*) – Pair coupling score used for calculation
- **min_seqdist** (*int, optional (default: 6)*) – Minimum sequence distance of couplings that will be included in the calculation

Returns **enrichment_table** – Sorted table with enrichment values for each position in the sequence

Return type pd.DataFrame

evcouplings.couplings.pairs.**read_raw_ec_file**(*filename*, *sort=True*, *score='cn'*)

Read a raw EC file (e.g. from plmc) and sort by scores

Parameters

- **filename** (*str*) – File containing evolutionary couplings
- **sort** (*bool, optional (default: True)*) – If True, sort pairs by coupling score in descending order
- **score_column** (*str, optional (default: True)*) – Score column to be used for sorting

Returns **ecs** – Table of evolutionary couplings

Return type pd.DataFrame

2.3.5 evcouplings.couplings.protocol module

2.3.6 evcouplings.couplings.tools module

Wrappers for tools for calculation of evolutionary couplings from sequence alignments.

Authors: Thomas A. Hopf

class evcouplings.couplings.tools.**PlmcResult**(*couplings_file*, *param_file*, *iteration_table*, *focus_seq_index*, *num_valid_seqs*, *num_total_seqs*, *num_valid_sites*, *num_total_sites*, *region_start*, *effective_samples*, *optimization_status*)

Bases: tuple

couplings_file
Alias for field number 0

effective_samples
Alias for field number 9

focus_seq_index
Alias for field number 3

iteration_table
Alias for field number 2

num_total_seqs
Alias for field number 5

num_total_sites
Alias for field number 7

num_valid_seqs
Alias for field number 4

num_valid_sites
Alias for field number 6

optimization_status
Alias for field number 10

param_file
Alias for field number 1

region_start
Alias for field number 8

evcouplings.couplings.tools.**parse_plmc_log**(*log*)
Parse plmc stderr text output into structured data

Parameters **log** (*str*) – stderr output from plmc

Returns

- **iter_df** (*pd.DataFrame*) – Table with iteration statistics
- **focus_index** (*int*) – Index of focus sequence in alignment
- **valid_seqs** (*int*) – Number of valid sequences in alignment
- **total_seqs** (*int*) – Number of total sequences in alignment
- **valid_sites** (*int*) – Analyzed number of sites in alignment/focus sequence

- **total_sites** (*int*) – Total number of sites in alignment/focus sequence
- **region_start** (*int*) – Index of first position in aligment
- **eff_samples** (*float*) – Effective number of samples in alignment
- **opt_status** (*str*) – End status of iterative optimization

evcouplings.couplings.tools.**run_plmc**(*alignment*, *couplings_file*, *param_file=None*, *focus_seq=None*, *alphabet=None*, *theta=None*, *scale=None*, *ignore_gaps=False*, *iterations=None*, *lambda_h=None*, *lambda_J=None*, *lambda_g=None*, *cpu=None*, *binary='plmc'*)

Run plmc on sequence alignment and store files with model parameters and pair couplings.

Parameters

- **alignment** (*str*) – Path to input sequence alignment
- **couplings_file** (*str*) – Output path for file with evolutionary couplings (folder will be created)
- **param_file** (*str*) – Output path for binary file containing model parameters (folder will be created)
- **focus_seq** (*str, optional (default: None)*) – Name of focus sequence, if None, non-focus mode will be used
- **alphabet** (*str, optional (default: None)*) – Alphabet for model inference. If None, standard amino acid alphabet including gap will be used. First character in string corresponds to gap character (relevant for ignore_gaps).
- **theta** (*float, optional (default: None)*) – Sequences with pairwise identity >= theta will be clustered and their sequence weights downweighted as 1 / num_cluster_members. Important: Note that plmc will be parametrized using 1 - theta. If None, default value in plmc will be used, which corresponds to theta=0.8 (plmc setting 0.2).
- **scale** (*float, optional (default: None)*) – Scale weights of clusters by this value. If None, default value in plmc (1.0) will be used
- **ignore_gaps** (*bool, optional (default: False)*) – Exclude gaps from parameter inference. Gap character is first character of alphabet parameter.
- **iterations** (*int, optional (default: None)*) – Maximum iterations for optimization.
- **lambda_h** (*float, optional (default: None)*) – l2 regularization strength on fields. If None, plmc default will be used.
- **lambda_J** (*float, optional (default: None)*) – l2-regularization strength on couplings. If None, plmc default will be used
- **lambda_g** (*float, optional (default: None)*) – group l1-regularization strength on couplings If None, plmc default will be used.
- **cpu** (*Number of cores to use for running plmc.*) – Note that plmc has to be compiled in openmp mode to runnable with multiple cores. Can also be set to "max".
- **binary** (*str, optional (default: "plmc")*) – Path to plmc binary

Returns namedtuple containing output files and parsed fields from console output of plmc

Return type *PlmcResult*

Raises `ExternalToolError`

2.4 evcouplings.mutate package

2.4.1 evcouplings.mutate.calculations module

High-level mutation calculation functions for EVmutation

Todo: implement segment handling

Authors: Thomas A. Hopf Anna G. Green (generalization for multiple segments)

`evcouplings.mutate.calculations.`**`extract_mutations`**(*mutation_string*, *offset=0*, *sep=','*)

Turns a string containing mutations of the format I100V into a list of tuples with format (100, 'I', 'V') (index, from, to)

Parameters

- **`mutation_string`** (*`str`*) – Comma-separated list of one or more mutations (e.g. "K50R,I100V")
- **`offset`** (*`int, default: 0`*) – Offset to be added to the index/position of each mutation
- **`sep`** (*`str, default ","`*) – String used to separate multiple mutations

Returns List of tuples of the form (index+offset, from, to)

Return type list of tuples

`evcouplings.mutate.calculations.`**`predict_mutation_table`**(*model*, *table*, *output_column='prediction_epistatic'*, *mutant_column='mutant'*, *hamiltonian='full'*, *segment=None*)

Predicts all mutants in a dataframe and adds predictions as a new column.

If mutant_column is None, the dataframe index is used, otherwise the given column.

Mutations which cannot be calculated (e.g. not covered by alignment, or invalid substitution) using object are set to NaN.

Parameters

- **`model`** (`CouplingsModel`) – CouplingsModel instance used to compute mutation effects
- **`table`** (*`pandas.DataFrame`*) – DataFrame with mutants to which delta of statistical energy will be added
- **`mutant_column`** (*`str`*) – Name of column in table that contains mutants
- **`output_column`** (*`str`*) – Name of column in returned dataframe that will contain computed effects
- **`hamiltonian`** (*`{"full", "couplings", "fields"},`*) – default: "full" Use full Hamiltonian of exponential model (default), or only couplings / fields for statistical energy calculation.

- **segment** (*str, default: None*) – Specificy a segment identifier to use for the positions in the mutation table. This will only be used if the mutation table doesn't already have a segments column.

Returns Dataframe with added column (mutant_column) that contains computed mutation effects

Return type pandas.DataFrame

evcouplings.mutate.calculations.**single_mutant_matrix**(*model*, *output_column='prediction_epistatic'*, *exclude_self_subs=True*)

Create table with all possible single substitutions of target sequence in CouplingsModel object.

Parameters

- **model** (CouplingsModel) – Model that will be used to predict single mutants
- **output_column** (*str, default: "prediction_epistatic"*) – Name of column in Dataframe that will contain predictions
- **exclude_self_subs** (*bool, default: True*) – Exclude self-substitutions (e.g. A100A) from results

Returns DataFrame with predictions for all single mutants

Return type pandas.DataFrame

evcouplings.mutate.calculations.**split_mutants**(*x*, *mutant_column='mutant'*)

Splits mutation strings into individual columns in DataFrame (wild-type symbol(s), position(s), substitution(s), number of mutations). This function is e.g. helpful when computing average effects per position using pandas groupby() operations

Parameters

- **x** (*pandas.DataFrame*) – Table with mutants
- **mutant_column** (*str, default: "mutant"*) – Column which contains mutants, set to None to use index of DataFrame

Returns DataFrame with added columns "num_subs", "pos", "wt" and "subs" that contain the number of mutations, and split mutation strings (if higher-order mutations, symbols/numbers are comma-separated)

Return type pandas.DataFrame

2.4.2 evcouplings.mutate.protocol module

Sequence statistical energy and mutation effect computation protocols

Authors: Thomas A. Hopf Anna G. Green (complex)

evcouplings.mutate.protocol.**complex**(***kwargs*)

Protocol: Mutation effect prediction and visualization for protein complexes

Parameters **kwargs arguments** (*Mandatory*) – See list below in code where calling check_required

Returns

outcfg – Output configuration of the pipeline, including the following fields:

- mutation_matrix_file
- [mutation_dataset_predicted_file]

Return type dict

evcouplings.mutate.protocol.**run**(***kwargs*)

Run mutation protocol

Parameters **kwargs arguments** (*Mandatory*) – protocol: EC protocol to run prefix: Output prefix for all generated files

Returns **outcfg** – Output configuration of stage (see individual protocol for fields)

Return type dict

evcouplings.mutate.protocol.**standard**(***kwargs*)

Protocol: Mutation effect calculation and visualization for protein monomers

TODO: eventually merge with complexes to make a protocol agnostic to the number of segments

Parameters **kwargs arguments** (*Mandatory*) – See list below in code where calling check_required

Returns

outcfg – Output configuration of the pipeline, including the following fields:

- mutation_matrix_file
- [mutation_dataset_predicted_file]

Return type dict

CHAPTER 3

Folding Analysis

3.1 evcouplings.fold package

3.1.1 evcouplings.fold.cns module

3.1.2 evcouplings.fold.filter module

Functions for detecting ECs that should not be included in 3D structure prediction

Most functions in this module are rewritten from older pipeline code in choose_CNS_constraint_set.m

Authors: Thomas A. Hopf

evcouplings.fold.filter.**detect_secstruct_clash**(*i*, *j*, *secstruct*)

Detect if an EC pair (i, j) is geometrically impossible given a predicted secondary structure

Based on direct port of the logic implemented in choose_CNS_constraint_set.m from original pipeline, lines 351-407.

Use secstruct_clashes() to annotate an entire table of ECs.

Parameters

- **i** (*int*) – Index of first position
- **j** (*int*) – Index of second position
- **secstruct** (*dict*) – Mapping from position (int) to secondary structure ("H", "E", "C")

Returns **clashes** – True if (i, j) clashes with secondary structure

Return type bool

evcouplings.fold.filter.**disulfide_clashes**(*ec_pairs*, *output_column='cys_clash'*)

Add disulfide bridge clashes to EC table (i.e. if any cysteine residue is coupled to another cysteine). This flag is necessary if disulfide bridges are created during folding, since only one bridge is possible per cysteine.

Parameters

- **ec_pairs** (*pandas.DataFrame*) – Table with EC pairs that will be tested for the occurrence of multiple cys-cys pairings (with columns i, j, A_i, A_j)
- **output_column** (*str, optional (default: "cys_clash")*) – Target column indicating if pair is in a clash or not

Returns Annotated EC table with clashes

Return type pandas.DataFrame

evcouplings.fold.filter.**secstruct_clashes**(*ec_pairs*, *residues*, *output_column='ss_clash'*, *secstruct_column='sec_struct_3state'*)

Add secondary structure clashes to EC table

Parameters

- **ec_pairs** (*pandas.DataFrame*) – Table with EC pairs that will be tested for clashes with secondary structure (with columns i, j)
- **residues** (*pandas.DataFrame*) – Table with residues in sequence and their secondary structure (columns i, ss_pred).
- **output_column** (*str, optional (default: "secstruct_clash")*) – Target column indicating if pair is in a clash or not
- **secstruct_column** (*str, optional (default: "sec_struct_3state")*) – Source column in ec_pairs with secondary structure states (H, E, C)

Returns Annotated EC table with clashes

Return type pandas.DataFrame

3.1.3 evcouplings.fold.protocol module

3.1.4 evcouplings.fold.ranking module

3.1.5 evcouplings.fold.restraints module

Functions for generating distance restraints from evolutionary couplings and secondary structure predictions

Authors: Thomas A. Hopf Anna G. Green (docking restraints)

evcouplings.fold.restraints.**docking_restraints**(*ec_pairs*, *output_file*, *restraint_formatter*, *config_file=None*)

Create .tbl file with distance restraints for docking

Parameters

- **ec_pairs** (*pandas.DataFrame*) – Table with EC pairs that will be turned into distance restraints (with columns i, j, A_i, A_j, segment_i, segment_j)
- **output_file** (*str*) – Path to file in which restraints will be saved
- **restraint_formatter** (*function*) – Function called to create string representation of restraint
- **config_file** (*str, optional (default: None)*) – Path to config file with folding settings. If None, will use default settings included in package (restraints.yml).

evcouplings.fold.restraints.**ec_dist_restraints**(*ec_pairs*, *output_file*, *restraint_formatter*, *config_file=None*)

Create .tbl file with distance restraints based on evolutionary couplings

Logic based on choose_CNS_constraint_set.m, lines 449-515

Parameters

- **ec_pairs** (*pandas.DataFrame*) – Table with EC pairs that will be turned into distance restraints (with columns i, j, A_i, A_j)
- **output_file** (*str*) – Path to file in which restraints will be saved
- **restraint_formatter** (*function*) – Function called to create string representation of restraint
- **config_file** (*str, optional (default: None)*) – Path to config file with folding settings. If None, will use default settings included in package (restraints.yml).

evcouplings.fold.restraints.**secstruct_angle_restraints**(*residues*, *output_file*, *restraint_formatter*, *config_file=None*, *secstruct_column='sec_struct_3state'*)

Create .tbl file with dihedral angle restraints based on secondary structure prediction

Logic based on make_cns_angle_constraints.pl

Parameters

- **residues** (*pandas.DataFrame*) – Table containing positions (column i), residue type (column A_i), and secondary structure for each position
- **output_file** (*str*) – Path to file in which restraints will be saved
- **restraint_formatter** (*function, optional*) – Function called to create string representation of restraint
- **config_file** (*str, optional (default: None)*) – Path to config file with folding settings. If None, will use default settings included in package (restraints.yml).
- **secstruct_column** (*str, optional (default: sec_struct_3state)*) – Column name in residues dataframe from which secondary structure will be extracted (has to be H, E, or C).

evcouplings.fold.restraints.**secstruct_dist_restraints**(*residues*, *output_file*, *restraint_formatter*, *config_file=None*, *secstruct_column='sec_struct_3state'*)

Create .tbl file with distance restraints based on secondary structure prediction

Logic based on choose_CNS_constraint_set.m, lines 519-1162

Parameters

- **residues** (*pandas.DataFrame*) – Table containing positions (column i), residue type (column A_i), and secondary structure for each position
- **output_file** (*str*) – Path to file in which restraints will be saved
- **restraint_formatter** (*function*) – Function called to create string representation of restraint
- **config_file** (*str, optional (default: None)*) – Path to config file with folding settings. If None, will use default settings included in package (restraints.yml).

- **secstruct_column** (*str, optional (default: sec_struct_3state)*) – Column name in residues dataframe from which secondary structure will be extracted (has to be H, E, or C).

3.1.6 evcouplings.fold.tools module

Wrappers for tools for 3D structure prediction from evolutionary couplings

Authors: Thomas A. Hopf

evcouplings.fold.tools.**parse_maxcluster_clustering**(*clustering_output*)
Parse maxcluster clustering output into a DataFrame

Parameters **clustering_output** (*str*) – stdout output from maxcluster after clustering

Returns Parsed result table (columns: filename, cluster, cluster_size)

Return type pandas.DataFrame

evcouplings.fold.tools.**parse_maxcluster_comparison**(*comparison_output*)
Parse maxcluster output into a DataFrame

Parameters **comparison_output** (*str*) – stdout output from maxcluster after comparison

Returns Parsed result table (columns: filename, num_pairs, rmsd, maxsub, tm, msi), refer to maxcluster documentation for explanation of the score fields.

Return type pandas.DataFrame

evcouplings.fold.tools.**read_psipred_prediction**(*filename*, *first_index=1*)
Read a psipred secondary structure prediction file in horizontal or vertical format (auto-detected).

Parameters

- **filename** (*str*) – Path to prediction output file
- **first_index** (*int, optional (default: 1)*) – Index of first position in predicted sequence

Returns

pred – Table containing secondary structure prediction, with the following columns:

- i: position
- A_i: amino acid
- sec_struct_3state: prediction (H, E, C)

If reading vformat, also contains columns for the individual (score_coil/helix/strand)

If reading hformat, also contains confidence score between 1 and 9 (sec_struct_conf)

Return type pandas.DataFrame

evcouplings.fold.tools.**run_cns**(*inp_script=None*, *inp_file=None*, *log_file=None*, *binary='cns'*)
Run CNSsolve 1.21 (without worrying about environment setup)

Note that the user is responsible for verifying the output products of CNS, since their paths are determined by .inp scripts and hard to check automatically and in a general way.

Either input_script or input_file has to be specified.

Parameters

- **inp_script** (*str, optional (default: None)*) – CNS ".inp" input script (actual commands, not file)
- **inp_file** (*str, optional (default: None)*) – Path to .inp input script file. Will override inp_script if also specified.
- **log_file** (*str, optional (default: None)*) – Save CNS stdout output to this file
- **binary** (*str, optional (default: "cns")*) – Absolute path of CNS binary

Raises

- ExternalToolError – If call to CNS fails
- InvalidParameterError – If no input script (file or string) given

evcouplings.fold.tools.**run_cns_13**(*inp_script=None*, *inp_file=None*, *log_file=None*, *source_script=None*, *binary='cns'*)

Run CNSsolve 1.3

Note that the user is responsible for verifying the output products of CNS, since their paths are determined by .inp scripts and hard to check automatically and in a general way.

Either input_script or input_file has to be specified.

Parameters

- **inp_script** (*str, optional (default: None)*) – CNS ".inp" input script (actual commands, not file)
- **inp_file** (*str, optional (default: None)*) – Path to .inp input script file. Will override inp_script if also specified.
- **log_file** (*str, optional (default: None)*) – Save CNS stdout output to this file
- **source_script** (*str, optional (default: None)*) – Script to set CNS environment variables. This should typically point to .cns_solve_env_sh in the CNS installation main directory (the shell script itself needs to be edited to contain the path of the installation)
- **binary** (*str, optional (default: "cns")*) – Name of CNS binary

Raises

- ExternalToolError – If call to CNS fails
- InvalidParameterError – If no input script (file or string) given

evcouplings.fold.tools.**run_maxcluster_cluster**(*predictions*, *method='average'*, *rmsd=True*, *clustering_threshold=None*, *binary='maxcluster'*)

Compare a set of predicted structures to an experimental structure using maxcluster.

For clustering functionality, use run_maxcluster_clustering() function.

Parameters

- **predictions** (*list(str)*) – List of PDB files that should be compared against experiment
- **method** ({*"single", "average", "maximum", "pairs_min", "pairs_abs"}, optional (default: "average")*) – Clustering method (single / average / maximum linkage, or min / absolute size neighbour pairs

- **clustering_threshold** (*float (optional, default: None)*) – Initial clustering threshold (maxcluster -T option)
- **rmsd** (*bool, optional (default: True)*) – Use RMSD-based clustering (faster)
- **binary** (*str, optional (default: "maxcluster")*) – Path to maxcluster binary

Returns Clustering result table (see parse_maxcluster_clustering for more detailed explanation)

Return type pandas.DataFrame

evcouplings.fold.tools.**run_maxcluster_compare**(*predictions*, *experiment*, *normalization_length=None*, *distance_cutoff=None*, *binary='maxcluster'*)

Compare a set of predicted structures to an experimental structure using maxcluster.

For clustering functionality, use run_maxcluster_clustering() function.

For a high-level wrapper around this function that removes problematic atoms and compares multiple models, please look at evcouplings.fold.protocol.compare_models_maxcluster().

Parameters

- **predictions** (*list(str)*) – List of PDB files that should be compared against experiment
- **experiment** (*str*) – Path of experimental structure PDB file. Note that the numbering and residues in this file must agree with the predicted structure, and that the structure may not contain duplicate atoms (multiple models, or alternative locations for the same atom).
- **normalization_length** (*int, optional (default: None)*) – Use this length to normalize the Template Modeling (TM) score (-N option of maxcluster). If None, will normalize by length of experiment.
- **distance_cutoff** (*float, optional (default: None)*) – Distance cutoff for MaxSub search (-d option of maxcluster). If None, will use maxcluster autocalibration.
- **binary** (*str, optional (default: "maxcluster")*) – Path to maxcluster binary

Returns Comparison result table (see parse_maxcluster_comparison for more detailed explanation)

Return type pandas.DataFrame

evcouplings.fold.tools.**run_psipred**(*fasta_file*, *output_dir*, *binary='runpsipred'*)

Run psipred secondary structure prediction

psipred output file convention: run_psipred creates output files <rootname>.ss2 and <rootname2>.horiz in the current working directory, where <rootname> is extracted from the basename of the input file (e.g. /home/test/<rootname>.fa)

Parameters

- **fasta_file** (*str*) – Input sequence file in FASTA format
- **output_dir** (*str*) – Directory in which output will be saved
- **binary** (*str, optional (default: "cns")*) – Path of psipred executable (runpsipred)

Returns

- **ss2_file** (*str*) – Absolute path to prediction output in "VFORMAT"
- **horiz_file** (*str*) – Absolute path to prediction output in "HFORMAT"

Raises `ExternalToolError` – If call to psipred fails

CHAPTER 4

Visualization

4.1 evcouplings.visualize package

4.1.1 evcouplings.visualize.misc module

4.1.2 evcouplings.visualize.mutations module

4.1.3 evcouplings.visualize.pairs module

4.1.4 evcouplings.visualize.parameters module

Visualization of pair model parameters

Authors: Thomas A. Hopf

evcouplings.visualize.parameters.**evzoom_data**(*model*, *ec_threshold=0.9*, *freq_threshold=0.01*, *Jij_threshold=10*, *score='cn'*, *reorder='KRHEDNQTSCGAVLIMPYFW'*)

Generate data for EVzoom visualization. Use evzoom_json() to get final JSON string to use with EVzoom.

Parameters

- **model** (evcouplings.couplings.model.CouplingsModel) – Parameters of pairwise graphical model
- **ec_threshold** (*float or int, optional (default: 0.9)*) – Only display evolutionary couplings above this threshold. If float between 0 and 1, this will be interpreted as probability cutoff for mixture model. Otherwise, will be interpreted as absolute number of couplings.
- **freq_threshold** (*float, optional (default: 0.01)*) – Only display coupling parameters for amino acids with at least this frequency in the underlying sequence alignment

- **Jij_threshold** (*int or float, optional (default: 10)*) – Only display coupling parameters above this threshold. If float, this number will be interpreted as an actual score threshold; if int, this will be interpreted as a percentage of the maximum absolute score.
- **score** (*str, optional (default: "cn")*) – Use this score to determine which couplings to display. Valid choices are the score columns contained in the CouplingsModel.ecs dataframe
- **reorder** (*str, optional (default: "KRHEDNQTSCGAVLIMPYFW")*) – Order of amino acids in displayed coupling matrices

Returns

- **map_** (*dict*) – Map containing sequence indices and characters
- **logo** (*list*) – List containing information about sequence logos for axes of visualization
- **matrix** (*dict*) – List containing couplings that will be visualized

evcouplings.visualize.parameters.**evzoom_json**(*model*, ***kwargs*)

Create JSON input for visualizing parameters of pairwise graphical model using EVzoom

Parameters

- **model** (evcouplings.couplings.model.CouplingsModel) – Parameters of pairwise graphical model
- ****kwargs** – See evzoom_data() for options

Returns EVzoom-ready JSON input

Return type str

4.1.5 evcouplings.visualize.pymol module

CHAPTER 5

Utilities

5.1 evcouplings.utils package

5.1.1 evcouplings.utils.app module

5.1.2 evcouplings.utils.batch module

Looping through batches of jobs (former submit_job.py and buildali loop)

Authors: Benjamin Schubert, Thomas A. Hopf

class evcouplings.utils.batch.**AClusterSubmitter**

Bases: *evcouplings.utils.batch.ASubmitter*

Abstract subclass of a cluster submitter

cancel(*command*)

Consumes a list of jobIDs and trys to cancel them

Parameters **command** (Command) – The Command jobejct to cancel

Returns If job was canceled

Return type bool

cancel_command

db

The persistent DB to keep track of all submitted jobs and their status

Returns The Persistent DB

Return type *PersistentDict*

job_id_pattern

join()

Blocks script if so desired until all jobs have be finished canceled or died

monitor(*command*)
Returns the status of the consumed command

Parameters **command** (Command) – The command object whose status is inquired

Returns The status of the Command

Return type Enum(Status)

monitor_command

resource_flags

submit(*command*, *dependent=None*)
Consumes job objects and starts them

Parameters

- **jobs** (Command) – A list of Job objects that should be submitted
- **dependent** (*list(Command)*) – A list of command objects the Command depend on

Returns A list of jobIDs

Return type list(str)

submit_command()

class evcouplings.utils.batch.**APluginRegister**(*name*, *bases*, *nmspc*)
Bases: abc.ABCMeta

This class allows automatic registration of new plugins.

class evcouplings.utils.batch.**ASubmitter**
Bases: object

Interface for all submitters

cancel(*command*)
Consumes a list of jobIDs and trys to cancel them

Parameters **command** (Command) – The Command jobejct to cancel

Returns If job was canceled

Return type bool

isBlocking
Indicator whether the submitter is blocking or not

Returns whether submitter blocks by calling join or not

Return type bool

join()
Blocks script if so desired until all jobs have be finished canceled or died

monitor(*command*)
Returns the status of the consumed command

Parameters **command** (Command) – The command object whose status is inquired

Returns The status of the Command

Return type Enum(Status)

name
The name of the submitter

Returns The name of the submitter

Return type str

registry = {'local': <class 'evcouplings.utils.batch.LocalSubmitter'>, 'lsf': <class

submit(*command*, *dependent=None*)
Consumes job objects and starts them

Parameters

- **jobs** (Command) – A list of Job objects that should be submitted
- **dependent** (*list (*Command*)*) – A list of command objects the Command depend on

Returns A list of jobIDs

Return type list(str)

class evcouplings.utils.batch.**Command**(*command*, *name=None*, *environment=None*, *workdir=None*, *resources=None*)
Bases: object

Wrapper around the command parameters needed to execute a script

class evcouplings.utils.batch.**EJob**
Bases: enum.Enum

An enumeration.

CANCEL = 2

MONITOR = 1

PID = 5

STOP = 3

SUBMIT = 0

UPDATE = 4

evcouplings.utils.batch.**EResource**
alias of evcouplings.utils.batch.Enum

evcouplings.utils.batch.**EStatus**
alias of evcouplings.utils.batch.Enum

class evcouplings.utils.batch.**LSFSubmitter**(*blocking=False*, *db_path=None*)
Bases: *evcouplings.utils.batch.AClusterSubmitter*

Implements an LSF submitter

cancel_command

db
The persistent DB to keep track of all submitted jobs and their status

Returns The Persistent DB

Return type *PersistentDict*

isBlocking
Indicator whether the submitter is blocking or not

Returns whether submitter blocks by calling join or not

Return type bool

job_id_pattern

monitor_command

name
The name of the submitter

Returns The name of the submitter

Return type str

resource_flags

submit_command

class evcouplings.utils.batch.**LocalSubmitter**(*blocking=True*, *db_path=None*, *ncpu=1*)
Bases: *evcouplings.utils.batch.ASubmitter*

cancel(*command*)
Consumes a list of jobIDs and trys to cancel them

Parameters **command** (Command) – The Command jobejct to cancel

Returns If job was canceled

Return type bool

isBlocking
Indicator whether the submitter is blocking or not

Returns whether submitter blocks by calling join or not

Return type bool

join()
Blocks script if so desired until all jobs have be finished canceled or died

monitor(*command*)
Returns the status of the consumed command

Parameters **command** (Command) – The command object whose status is inquired

Returns The status of the Command

Return type Enum(Status)

name
The name of the submitter

Returns The name of the submitter

Return type str

submit(*command*, *dependent=None*)
Consumes job objects and starts them

Parameters

- **jobs** (Command) – A list of Job objects that should be submitted
- **dependent** (*list*(Command)) – A list of command objects the Command depend on

Returns A list of jobIDs

Return type list(str)

class evcouplings.utils.batch.**SGESubmitter**(*blocking=False*, *db_path=None*)
Bases: *evcouplings.utils.batch.AClusterSubmitter*

Implements an LSF submitter

cancel_command

db
The persistent DB to keep track of all submitted jobs and their status

Returns The Persistent DB

Return type *PersistentDict*

isBlocking
Indicator whether the submitter is blocking or not

Returns whether submitter blocks by calling join or not

Return type bool

job_id_pattern

monitor_command

name
The name of the submitter

Returns The name of the submitter

Return type str

resource_flags

submit_command

class evcouplings.utils.batch.**SlurmSubmitter**(*blocking=False*, *db_path=None*)
Bases: *evcouplings.utils.batch.AClusterSubmitter*

Implements an LSF submitter

cancel_command

db
The persistent DB to keep track of all submitted jobs and their status

Returns The Persistent DB

Return type *PersistentDict*

isBlocking
Indicator whether the submitter is blocking or not

Returns whether submitter blocks by calling join or not

Return type bool

job_id_pattern

monitor_command

name
The name of the submitter

Returns The name of the submitter

Return type str

resource_flags

submit_command

5.1.3 evcouplings.utils.calculations module

General calculation functions.

Authors: Thomas A. Hopf

evcouplings.utils.calculations.**dihedral_angle**(*p0*, *p1*, *p2*, *p3*)
Compute dihedral angle given four points

Adapted from the following source: http://stackoverflow.com/questions/20305272/dihedral-torsion-angle-from-four-points-in-cartesian-coordinates-in-python (answer by user Praxeolitic)

Parameters

- **p0** (*np.array*) – Coordinates of first point
- **p1** (*np.array*) – Coordinates of second point
- **p2** (*np.array*) – Coordinates of third point
- **p3** (*np.array*) – Coordinates of fourth point

Returns Dihedral angle (in radians)

Return type numpy.float

evcouplings.utils.calculations.**entropy**(*X*, *normalize=False*)
Calculate entropy of distribution

Parameters

- **X** (*np.array*) – Vector for which entropy will be calculated
- **normalize** – Rescale entropy to range from 0 ("variable", "flat") to 1 ("conserved")

Returns Entropy of X

Return type float

evcouplings.utils.calculations.**entropy_map**(*model*, *normalize=True*)
Compute dictionary of positional entropies for single-site frequencies in a CouplingsModel

Parameters

- **model** (CouplingsModel) – Model for which entropy of sequence alignment will be computed (based on single-site frequencies f_i(A_i) contained in model)
- **normalize** (*bool, default: True*) – Normalize entropy to range 0 (variable) to 1 (conserved) instead of raw values

Returns Map from positions in sequence (int) to entropy of column (float) in alignment

Return type dict

evcouplings.utils.calculations.**entropy_vector**(*model*, *normalize=True*)
Compute vector of positional entropies for single-site frequencies in a CouplingsModel

Parameters

- **model** (CouplingsModel) – Model for which entropy of sequence alignment will be computed (based on single-site frequencies f_i(A_i) contained in model)

- **normalize** (*bool, default: True*) – Normalize entropy to range 0 (variable) to 1 (conserved) instead of raw values

Returns Vector of length model.L containing entropy for each position

Return type np.array

5.1.4 evcouplings.utils.config module

Configuration handling

Todo: switch ruamel.yaml to round trip loading to preserver order and comments?

Authors: Thomas A. Hopf

exception evcouplings.utils.config.**InvalidParameterError**
Bases: Exception

Exception for invalid parameter settings

exception evcouplings.utils.config.**MissingParameterError**
Bases: Exception

Exception for missing parameters

evcouplings.utils.config.**check_required**(*params*, *keys*)
Verify if required set of parameters is present in configuration

Parameters

- **params** (*dict*) – Dictionary with parameters
- **keys** (*list-like*) – Set of parameters that has to be present in params

Raises *MissingParameterError*

evcouplings.utils.config.**parse_config**(*config_str*, *preserve_order=False*)
Parse a configuration string

Parameters

- **config_str** (*str*) – Configuration to be parsed
- **preserve_order** (*bool, optional (default: True)*) – Preserve formatting of input configuration string

Returns Configuration dictionary

Return type dict

evcouplings.utils.config.**read_config_file**(*filename*, *preserve_order=False*)
Read and parse a configuration file.

Parameters **filename** (*str*) – Path of configuration file

Returns Configuration dictionary

Return type dict

evcouplings.utils.config.**write_config_file**(*out_filename*, *config*)
Save configuration data structure in YAML file.

Parameters

- **out_filename** (*str*) – Filename of output file
- **config** (*dict*) – Config data that will be written to file

5.1.5 evcouplings.utils.constants module

Useful values and constants for all of package

Authors: Thomas A. Hopf

5.1.6 evcouplings.utils.database module

Job status database models and handling

Authors: Thomas A. Hopf

evcouplings.utils.database.**EStatus**
alias of evcouplings.utils.database.Enum

evcouplings.utils.database.**update_job_status**(*config*, *status=None*, *stage=None*)
Update job status based on configuration and update request by pipeline

Parameters

- **config** (*dict-like*) – Job configuration dictionary. Accessed entries are {
  global : {prefix} management: {database_uri, job_name}
  }
- **status** (*str, optional (default: None)*) – If not None, update job status to this value
- **stage** (*str, optional (default: None)*) – If not None, update job stage to this value

5.1.7 evcouplings.utils.helpers module

Useful Python helpers

Authors: Thomas A. Hopf, Benjamin Schubert

class evcouplings.utils.helpers.**DefaultOrderedDict**(*default_factory=None*, ***kwargs*)
Bases: collections.OrderedDict

Source: http://stackoverflow.com/questions/36727877/inheriting-from-defaultddict-and-ordereddict Answer by http://stackoverflow.com/users/3555845/daniel

Maybe this one would be better? http://stackoverflow.com/questions/6190331/can-i-do-an-ordered-default-dict-in-python

class evcouplings.utils.helpers.**PersistentDict**(*filename*, *flag='c'*, *mode=None*, *format='json'*, **args*, ***kwds*)
Bases: dict

Persistent dictionary with an API compatible with shelve and anydbm.

The dict is kept in memory, so the dictionary operations run as fast as a regular dictionary.

Write to disk is delayed until close or sync (similar to gdbm's fast mode).

Input file format is automatically discovered. Output file format is selectable between pickle, json, and csv. All three serialization formats are backed by fast C implementations.

https://code.activestate.com/recipes/576642/

close()

dump(*fileobj*)

load(*fileobj*)

sync()
Write dict to disk

class evcouplings.utils.helpers.**Progressbar**(*total_size*, *bar_length=60*)
Bases: object

Progress bar for command line programs

Parameters

- **total_size** (*int*) – The total size of the iteration
- **bar_length** (*int*) – The visual bar length that gets printed on stdout

update(*chunk*)
Updates and prints the progress of the progressbar

Parameters **chunk** (*int*) – The size of the elements that are processed in the current iteration

evcouplings.utils.helpers.**range_overlap**(*a*, *b*)

Source: **http://stackoverflow.com/questions/2953967/** built-in-function-for-computing-overlap-in-python

Function assumes that start < end for a and b

Note: Ends of range are not inclusive

Parameters

- **a** (*tuple(int, int)*) – Start and end of first range (end of range is not inclusive)
- **b** (*tuple(int, int)*) – Start and end of second range (end of range is not inclusive)

Returns Length of overlap between ranges a and b

Return type int

evcouplings.utils.helpers.**render_template**(*template_file*, *mapping*)
Render a template using jinja2 and substitute values from mapping

Parameters

- **template_file** (*str*) – Path to jinja2 template
- **mapping** (*dict*) – Mapping used to substitute values in the template

Returns Rendered template

Return type str

evcouplings.utils.helpers.**wrap**(*text*, *width=80*)
Wraps a string at a fixed width.

Parameters

- **text** (*str*) – Text to be wrapped
- **width** (*int*) – Line width

Returns Wrapped string

Return type str

5.1.8 evcouplings.utils.pipeline module

5.1.9 evcouplings.utils.summarize module

5.1.10 evcouplings.utils.system module

System-level calls to external tools, directory creation, etc.

Authors: Thomas A. Hopf

exception evcouplings.utils.system.**ExternalToolError**
Bases: Exception

Exception for failing external calculations

exception evcouplings.utils.system.**ResourceError**
Bases: Exception

Exception for missing resources (files, URLs, ...)

evcouplings.utils.system.**create_prefix_folders**(*prefix*)
Create a directory tree contained in a prefix.

prefix [str] Prefix containing directory tree

evcouplings.utils.system.**get**(*url*, *output_path=None*, *allow_redirects=False*)
Download external resource

Parameters

- **url** (*str*) – URL of resource that should be downloaded
- **output_path** (*str, optional*) – Save contents of URL to this file (only for text files)
- **allow_redirects** (*bool*) – Allow redirects by server or not

Returns **r** – Response object, use r.text to access text, r.json() to decode json, and r.content for raw bytestring

Return type requests.models.Response

Raises *ResourceError*

evcouplings.utils.system.**get_urllib**(*url*, *output_path*)
Download external resource to file using urllib. This function is intended for cases where get() implemented using requests can not be used, e.g. for download from an FTP server.

Parameters

- **url** (*str*) – URL of resource that should be downloaded
- **output_path** (*str, optional*) – Save contents of URL to this file (only for text files)

evcouplings.utils.system.**insert_dir**(*prefix*, **dirs*, *rootname_subdir=True*)
Create new path by inserting additional directories into the folder tree of prefix (but keeping the filename prefix at the end),

Parameters

- **prefix** (*str*) – Prefix of path that should be extended
- ***dirs** (*str*) – Add these directories at the end of path
- **rootname_subdir** (*bool, optional (default: True)*) – Given /my/path/prefix,
  - if True, creates structure like /my/path/prefix/*dirs/prefix
  - if False, creates structure like /my/path/*dirs/prefix

Returns Extended path

Return type str

evcouplings.utils.system.**makedirs**(*directories*)
Create directory subtree, some or all of the folders may already exist.

Parameters **directories** (*str*) – Directory subtree to create

evcouplings.utils.system.**run**(*cmd*, *stdin=None*, *check_returncode=True*, *working_dir=None*, *shell=False*, *env=None*)
Run external program as subprocess.

Parameters

- **cmd** (*str or list of str*) – Command (and optional command line arguments)
- **stdin** (*str or byte sequence, optional (default: None)*) – Input to be sent to STDIN of the process
- **check_returncode** (*bool, optional (default=True)*) – Verify if call had returncode == 0, otherwise raise ExternalToolError
- **working_dir** (*str, optional (default: None)*) – Change to this directory before running command
- **shell** (*bool, optional (default: False)*) – Invoke shell when calling subprocess (default: False)
- **env** (*dict, optional (default: None)*) – Use this environment for executing the subprocess

Returns

- *int* – Return code of process
- *stdout* – Byte string with stdout output
- *stderr* – Byte string of stderr output

Raises *ExternalToolError*

evcouplings.utils.system.**temp**()
Create a temporary file

Returns Path of temporary file

Return type str

evcouplings.utils.system.**tempdir**()
Create a temporary directory

Returns Path of temporary directory

Return type str

evcouplings.utils.system.**valid_file**(*file_path*)
Verify if a file exists and is not empty.

Parameters **file_path** (*str*) – Path to file to check

Returns True if file exists and is non-zero size, False otherwise.

Return type bool

evcouplings.utils.system.**verify_resources**(*message*, **args*)
Verify if a set of files exists and is not empty.

Parameters

- **message** (*str*) – Message to display with raised ResourceError
- ***args** (*List of str*) – Path(s) of file(s) to be checked

Raises *ResourceError* – If any of the resources does not exist or is empty

evcouplings.utils.system.**write_file**(*file_path*, *content*)
Writes content to output file

Parameters

- **file_path** (*str*) – Path of output file
- **content** (*str*) – Content to be written to file

5.1.11 evcouplings.utils.update_database module

command-line app to update the necessary databases

Authors: Benjamin Schubert

evcouplings.utils.update_database.**download_ftp_file**(*ftp_url*, *ftp_cwd*, *file_url*, *output_path*, *file_handling='wb'*, *gziped=False*, *verbose=False*)
Downloads a gzip file from a remote ftp server and decompresses it on the fly into an output file

Parameters

- **ftp_url** (*str*) – the FTP server url
- **ftp_cwd** (*str*) – the FTP directory of the file to download
- **file_url** (*str*) – the file name that gets downloaded
- **output_path** (*str*) – the path to the output file on the local system
- **file_handling** (*str*) – the file handling mode (default: 'wb')
- **verbose** (*bool*) – determines whether a progressbar is printed

evcouplings.utils.update_database.**run**(***kwargs*)
Exposes command line interface as a Python function.

Parameters **kwargs** – See click.option decorators for app() function

evcouplings.utils.update_database.**symlink_force**(*target*, *link_name*)
Creates or overwrites an existing symlink

Parameters

- **target** (*str*) – the target file path
- **link_name** (*str*) – the symlink name

CHAPTER 6

Indices and tables

- genindex
- modindex
- search

Python Module Index

e

Index

Symbols

A

B

C

D

E

F

G

H

I

J

L

M

N

O

P

R

S

T

U

V

W