Jade Documentation

Release 1.0

Jared Adolf-Bryfogle

Jun 04, 2018

Contents

Contents:

CHAPTER 1

Bio-Jade

A repository for modules and applications to aid in the design and analysis of Biological molecules, especially when working with Rosetta or PyRosetta.

- Free software: BSD license
- Documentation: https://bio-jade.readthedocs.io.

1.1 Features

- A suite of modules for working with biological modules in python.
- A suite of public and pilot applications to make day-to-day tasks easier.
  - Commonly used ones include score_analysis.py, get_seq.py, and RunRosettaMPI.py

1.2 Rosetta

- Rosetta : http://www.rosettacommons.org
- PyRosetta: http://www.pyrosetta.org

1.3 Caveats

This package is still under heavy development, and test code coverage is limited at the moment.

1.4 Credits

This package was in part created with Cookiecutter and the audreyr/cookiecutter-pypackage project template.

CHAPTER 2

Installation

2.1 Stable release

To install Jade, run this command in your terminal:

```
$ pip install bio-jade
```

This is the preferred method to install Jade, as it will always install the most recent stable release.

If you don't have pip installed, this Python installation guide can guide you through the process.

2.2 From sources

The sources for Jade can be downloaded from the Github repo.

You can either clone the public repository:

```
$ git clone git://github.com/SchiefLab/Jade
```

Or download the tarball:

```
$ curl  -OL https://github.com/SchiefLab/Jade/tarball/master
```

Once you have a copy of the source, you can install it with:

```
$ pip install -e .
```

This will symlink your cloned code instead of copying it - very useful for development.

Don't forget to source your bashrc/bash_profile or whatever other shell profile you have.

2.2.1 Scripts

All scripts and applications in jade/apps will be installed in your bin directory and available for use.

2.3 Rosetta

- Rosetta : http://www.rosettacommons.org
- PyRosetta: http://www.pyrosetta.org

CHAPTER 3

API Documentation

3.1 Jade API

3.1.1 Subpackages

jade.antibody package

Subpackages

jade.antibody.decoy_data package

jade.antibody.decoy_data.DecoyData module

class jade.antibody.decoy_data.DecoyData.**DecoyData**(*name*, *has_real_values=True*, *reverse_top=False*)

Bases: object

add_data(*strategy*, *con*)
Baseclass method - needs to be overridden in subclass :param strategy: Strategy to which we are adding data. :param con: Sqlite3 Connection object

add_filters(*filters*, *filter_name*)

get_concatonated_map(*by_score_tuple=False*)
Returns a defaultDic: Default:

decoy: DecoyDataTriple

by_score_tuple (for sorting on score and having possible redundancy) [score, decoy]: DecoyDataTriple

get_data_for_decoy(*strategy*, *decoy*)
Get the held data for the decoy :param strategy: Strategy Name :param decoy: Decoy name (with dir and suffix) :rtype: DecoyDataTriple

get_ordered_decoy_list(*strategy*, *top_n=None*)
Get an ordered array of decoy names by energy for a particular strategy :rtype: list of str

get_ordered_decoy_list_all(*top_n=None*)
Get an ordered array of decoy names by energy over all the strategies :rtype: list of str

get_outname()

get_pandas_dataframe(*top_n=None*, *drop_dir_prfix=False*)
Gets all data as a pandas dataframe. Uses the set name as the score. You can then order, or select specific ones using the data frame. :return: pandas.DataFrame

get_strategy_data(*strategy*, *by_score_tuple=False*)
For a particular strategy: Return a dictionary of decoy:DataTriple or if by_score_tuple:

[score, decoy] = DataTriple

get_top_all_data(*top_n*, *by_score_tuple=False*)
Over all the strategies: Return a dictionary of decoy:DataTriple or if by_score_tuple:

[score, decoy] = DataTriple

get_top_strategy_data(*strategy*, *top_n*, *by_score_tuple=False*)
For a particular strategy: Return a dictionary of decoy:DataTriple or if by_score_tuple:

[score, decoy] = DataTriple

For only the top scoring decoys

get_top_x_percent_cutoff_value(*strategy*, *top_percent*)

has_real_values()

set_interface(*interface*)
Set the Antibody-Antigen interface - used mainly for H_A vs LH_A

class jade.antibody.decoy_data.DecoyData.**DecoyDataTriple**(*strategy*, *struct_id*, *decoy*, *score*, *out_name*, *raw_name*)
Bases: object

Struct for holding data instead of a tupple

jade.antibody.decoy_data.DecoyDataTypes module

class jade.antibody.decoy_data.DecoyDataTypes.**CombinedStrDecoyData**(*filters*, *filt_name*)
Bases: jade.antibody.decoy_data.DecoyData.DecoyData

DecoyData class that has value as a string of the 3 main scores. Value held in DecoyDataTriple is a string: dG::total::dSASA for reference.

add_data(*strategy*, *con*)
Baseclass method - needs to be overridden in subclass :param strategy: Strategy to which we are adding data. :param con: Sqlite3 Connection object

class jade.antibody.decoy_data.DecoyDataTypes.**DeltaUnsatsPerAreaDecoyData**
Bases: jade.antibody.decoy_data.DecoyData.DecoyData

add_data(*strategy*, *con*)
Baseclass method - needs to be overridden in subclass :param strategy: Strategy to which we are adding data. :param con: Sqlite3 Connection object

class jade.antibody.decoy_data.DecoyDataTypes.**IntHbondDecoyData**
Bases: jade.antibody.decoy_data.DecoyData.DecoyData

New way for int hbonds - added directly from IAM.

add_data(*strategy*, *con*)
Baseclass method - needs to be overridden in subclass :param strategy: Strategy to which we are adding data. :param con: Sqlite3 Connection object

class jade.antibody.decoy_data.DecoyDataTypes.**InterfaceHBondDecoyDataLoader**
Bases: jade.antibody.decoy_data.DecoyData.DecoyData

DecoyData class that holds the number of LH_A or L_H interface Hbonds, and energies. Very Slow to get this information.

- SO - Subsequent Hbond classes accept this on construction and then parse its information

add_data(*strategy*, *con*)
Baseclass method - needs to be overridden in subclass :param strategy: Strategy to which we are adding data. :param con: Sqlite3 Connection object

class jade.antibody.decoy_data.DecoyDataTypes.**InterfaceHbondCountDecoyData**
Bases: jade.antibody.decoy_data.DecoyData.DecoyData

add_data(*strategy*, *con*)
Baseclass method - needs to be overridden in subclass :param strategy: Strategy to which we are adding data. :param con: Sqlite3 Connection object

setup_from_loader(*hbond_loader*)

class jade.antibody.decoy_data.DecoyDataTypes.**InterfaceHbondEnergyDecoyData**
Bases: jade.antibody.decoy_data.DecoyData.DecoyData

add_data(*strategy*, *con*)
Baseclass method - needs to be overridden in subclass :param strategy: Strategy to which we are adding data. :param con: Sqlite3 Connection object

setup_from_loader(*hbond_loader*)

class jade.antibody.decoy_data.DecoyDataTypes.**SCValueDecoyData**
Bases: jade.antibody.decoy_data.DecoyData.DecoyData

add_data(*strategy*, *con*)
Baseclass method - needs to be overridden in subclass :param strategy: Strategy to which we are adding data. :param con: Sqlite3 Connection object

class jade.antibody.decoy_data.DecoyDataTypes.**TotalDecoyData**
Bases: jade.antibody.decoy_data.DecoyData.DecoyData

add_data(*strategy*, *con*)
Baseclass method - needs to be overridden in subclass :param strategy: Strategy to which we are adding data. :param con: Sqlite3 Connection object

class jade.antibody.decoy_data.DecoyDataTypes.**dGDecoyData**
Bases: jade.antibody.decoy_data.DecoyData.DecoyData

add_data(*strategy*, *con*)
Baseclass method - needs to be overridden in subclass :param strategy: Strategy to which we are adding data. :param con: Sqlite3 Connection object

class jade.antibody.decoy_data.DecoyDataTypes.**dGTotalScoreSubset**
Bases: jade.antibody.decoy_data.DecoyData.DecoyData

dG of the top x percent of total score (for each strategy)

add_data(*strategy*, *con*, *top_total_percent*)
Baseclass method - needs to be overridden in subclass :param strategy: Strategy to which we are adding data. :param con: Sqlite3 Connection object

class jade.antibody.decoy_data.DecoyDataTypes.**dSASADecoyData**
Bases: jade.antibody.decoy_data.DecoyData.DecoyData

add_data(*strategy*, *con*)
Baseclass method - needs to be overridden in subclass :param strategy: Strategy to which we are adding data. :param con: Sqlite3 Connection object

jade.antibody.CDRClusterer module

class jade.antibody.CDRClusterer.**CDRClusterer**(*bio_pose*)
A simple class for calculating a CDRs cluster from dihedrals or a renumbered pose.

get_fullcluster(*cdr_name*, *chain=None*, *region=None*)
IF DIHEDRALS is SET - AKA from before using the same class - WILL USE THE SAME DIHEDRALS AS BEFORE Rewritten from C++ code. Identifies the cluster of a known cdr type given either custom dihedrals or dihedrals calculated from a pose. Returns a pair or [cdr_cluster, distance] region is [int start, int end, chain] - This way you can cluster without renumbering if you want.

Return type list[str, float]

get_length(*cdr_name*)

set_custom_dihedrals(*dihedrals*)
Dihedrals is a dict: ['phi']=[x, y, z]; ['psi'] = [x, y, z]; ['omega'] = [x, y, z (degrees)]

set_dihedrals_from_bio_pose(*start*, *end*, *chain*)
Set dihedrals from a BioPose :param start: int :param end: int :param chain: str

set_dihedrals_from_cdr(*cdr_name*, *chain*)

jade.antibody.ClusterData module

class jade.antibody.ClusterData.**CDRClusterData**(*ab_db*, *cdr_dir*, *limit_to_known=True*)
Bases: jade.antibody.ClusterData.Data

get_cdr_names()

get_cdr_paths()
Get the CDR path using the cdr directory

get_center_data()

get_center_name()

get_center_path()

get_data()

get_extra_data()

get_infos()

get_original_chain()

get_pdb()

has_center_data()

Returns boolean

load_data(*cdr*, *length*, *cluster*, *extra_sele=[]*)

class jade.antibody.ClusterData.**CDRClusters**(*ab_db*, *cdr_dir*, *limit_to_known=True*)
Bases: jade.antibody.ClusterData.Data

get_cluster_data(*cluster*)

Parameters **cluster** – string

Returns CDRClusterData

get_clusters()

load_data(*cdr*, *length*, *extra_sele=[]*)

class jade.antibody.ClusterData.**CDRData**(*ab_db*, *cdr_dir*, *limit_to_known=True*)
Bases: jade.antibody.ClusterData.Data

get_all_data()

get_cluster_data(*cdr_name*, *length*, *cluster*)

Parameters

- **cdr_name** – string
- **length** – int
- **cluster** – string

Returns CDRClusterData

get_clusters(*cdr_name*, *length*)

get_clusters_data(*cdr_name*, *length*)

Parameters

- **cdr_name** – string
- **length** – int

Returns CDRClusters

get_lengths(*cdr_name*)

Parameters **cdr_name** – string

Returns list

get_lengths_data(*cdr_name*)

Parameters **cdr_name** – string

Returns CDRLengths

load_data(*extra_sele=[]*)

class jade.antibody.ClusterData.**CDRLengths**(*ab_db*, *cdr_dir*, *limit_to_known=True*)
Bases: jade.antibody.ClusterData.Data

get_clusters_data(*length*)

Parameters **length** – int

Returns CDRCLusters

get_lengths()

load_data(*cdr*, *extra_sele=[]*)

class jade.antibody.ClusterData.**Data**(*ab_db*, *cdr_dir*, *limit_to_known=True*)

load_data(*extra_sele=''*)

set_res_cutoff(*res_cutoff*)

set_rfac_cutoff(*rfac_cutoff*)

class jade.antibody.ClusterData.**Info**(*path*, *name*, *PDB*, *original_chain*, *cluster*, *dihedral_distance*)

jade.antibody.ab_db module

jade.antibody.ab_db.**get_all_clusters_for_length**(*db*, *cdr*, *length*, *limit_to_known=True*, *res_cutoff=2.8*, *rfac_cutoff=0.3*)

Get all unique clusters for a length and a cdr

jade.antibody.ab_db.**get_all_lengths**(*db*, *cdr*, *limit_to_known=True*, *res_cutoff=2.8*, *rfac_cutoff=0.3*)

Get all unique lengths for a CDR

jade.antibody.ab_db.**get_cdr_rmsd_for_entry**(*db*, *pdb*, *original_chain*, *cdr*, *length*, *fullcluster*)

jade.antibody.ab_db.**get_center_dih_degrees_for_cluster_and_length**(*db*, *cdr*, *length*, *cluster*)

Returns a dictionary of center dihedral angles in positional order. Or returns False if not found. result["phis'] = [phis as floats] result["psis"] = [Psis as floats] result["omegas"] = [Omegas as floats]

jade.antibody.ab_db.**get_center_for_cluster_and_length**(*db*, *cdr*, *length*, *cluster*, *data_names_array*)

Get the center for a particular cluster and length

jade.antibody.ab_db.**get_data_for_cluster_and_length**(*db*, *cdr*, *length*, *cluster*, *data_names_array*, *limit_to_known=True*, *res_cutoff=2.8*, *rfac_cutoff=0.3*)

Get a set of data of a particular length, cdr, and cluster. data_names_array is a list of the types of data. Can include DISTINCT keyword

Example: data_names_array = ["PDB", "original_chain", "new_chain", "sequence"]

jade.antibody.ab_db.**get_dihedral_string_for_centers**(*db*, *limit_to_known=True*)

Get a string of the dihedral angles for all centers

jade.antibody.ab_db.**get_pdb_chain_subset**(*db*, *gene*, *use_cutoffs=False*, *res_cutoff=2.8*, *rfac_cutoff=0.3*)

Return a list of tuples of [pdb, chain] of the particular gene

jade.antibody.ab_db.**get_stem_rmsd_for_entry**(*db*, *pdb*, *original_chain*, *cdr*, *length*, *fullcluster*)

jade.antibody.ab_db.**get_unique_sequences_for_cluster**(*db*, *cluster*, *include_outliers*, *outlier_definition='conservative'*)

jade.antibody.outliers module

jade.antibody.outliers.**get_outlier_definition_string**(*outlier_definition*, *rmsd_cutoff=1.5*, *dihdist_cutoff=40*)

Returns a string for adding to a database query which removes outliers. Need to add AND manually to the string.

jade.antibody.outliers.**get_outlier_string**(*include_outliers*, *outlier_definition*, *add_AND=True*)

jade.antibody.split_structure module

jade.antibody.split_structure.**has_Fc**(*parent_PDB*)

jade.antibody.split_structure.**has_chain**(*pdb_map*, *chain*)

jade.antibody.split_structure.**message**(*pdb_name*, *type*)

jade.antibody.split_structure.**run_main**(*ab_dir*, *output_dir*, *only_dimer=True*)

jade.antibody.split_structure.**run_split_proto_CDR4**(*ab_dir*, *output_dir*, *overhang=0*, *skip_present=False*)

jade.antibody.split_structure.**run_split_proto_CDR4_by_gene**(*db*, *ab_dir*, *output_dir*, *overhang=0*, *skip_present=False*, *res_cutoff=2.8*, *rfac_cutoff=0.3*)

jade.antibody.split_structure.**separate_pdb**(*pdb_path*, *output_dir*, *only_dimer=True*)

Determine if we have Fv or FAB. If FAB, split into parts: Fc, Fv, linker.

jade.antibody.split_structure.**separate_proto_CDR4**(*pdb_path*, *output_dir*, *chain*, *overhang=0*, *skip_present=True*)

jade.antibody.split_structure.**split_CDR4**(*parent_PDB*, *chain*, *overhang=0*)

jade.antibody.split_structure.**split_Fc**(*parent_PDB*)

Split Fc from FAB. Return new pdb_map to save

jade.antibody.split_structure.**split_Fv**(*parent_PDB*)

Split Fv from FAB. Return new pdb_map to save

jade.antibody.split_structure.**split_linker**(*parent_PDB*)

Split 4 Residue linker from FAB. Linker may be longer than this, but I think this is about it.

jade.antibody.split_structure.**split_linker_H**(*parent_PDB*)

Split 4 Residue linker from FAB. Linker may be longer than this, but I think this is about it.

jade.antibody.split_structure.**split_linker_L**(*parent_PDB*)

Split 4 Residue linker from FAB. Linker may be longer than this, but I think this is about it.

jade.antibody.util module

jade.antibody.util.**get_all_cdr_sele**(*cdr*, *stem=0*)
If any stem residues are given, will include stem residues in selection.

jade.antibody.util.**get_overhang_sele**(*pymol_writer*, *cdr*, *overhang=3*)
Gets the selection for terminal superposition in PyMol

jade.basic package

Subpackages

jade.basic.TKinter package

jade.basic.TKinter.ImageFrame module

class jade.basic.TKinter.ImageFrame.**ImageFrame**(*_tk_*, *file_path*, ***options*)
Bases: Tkinter.Frame

jade.basic.TKinter.Listbox module

class jade.basic.TKinter.Listbox.**AutoListbox**(*master=None*, *cnf={}*, ***kw*)
Bases: Tkinter.Listbox

autowidth(*maxwidth*, *list=None*)

jade.basic.filters package

jade.basic.filters.DataFilter module

class jade.basic.filters.DataFilter.**DataFilter**(*name*, *type='boolean'*)
Class for storing filter information - such as not equal to values or cutoffs. Used to create custom queries for data.

get_required_tables()

get_required_wheres()

jade.basic.filters.DataFilters module

class jade.basic.filters.DataFilters.**H3ExtendedFilter**
Bases: jade.basic.filters.DataFilter.DataFilter

Filter to remove kinked H3 structures

class jade.basic.filters.DataFilters.**TotalScoreCutoffFilter**(*value*)
Bases: jade.basic.filters.DataFilter.DataFilter

Filter to remove structures with total_score greater than a particular value

set_value(*value*)

class jade.basic.filters.DataFilters.**dGCutoffFilter**(*value*)
Bases: jade.basic.filters.DataFilter.DataFilter

Filter to remove structures with LH_A dG greater than a particular value

set_value(*value*)

class jade.basic.filters.DataFilters.**dSASACutoffFilter**(*value*)
Bases: jade.basic.filters.DataFilter.DataFilter

Filter to remove dSASA greater than some value

set_value(*value*)

jade.basic.filters.FilterSettings module

class jade.basic.filters.FilterSettings.**FilterSettings**
Simple class for accessing energy cutoff settings for custom lists made using cutoffs.

get_energy_cutoff(*energy_type*)

get_energy_enabled(*energy_type*)

set_energy_cutoff(*energy_type*, *setting*)

set_energy_enabled(*energy_type*, *setting*)

jade.basic.pandas package

jade.basic.pandas.PandasDataFrame module

class jade.basic.pandas.PandasDataFrame.**GeneralPandasDataFrame**(*data=None*, *index=None*, *columns=None*, *dtype=None*, *copy=False*)
Bases: pandas.core.frame.DataFrame

detect_numeric()

drop_duplicate_columns()
Drop Duplicate columns from the DataFrame in place :return:

get_columns(*columns*)

get_matches(*column*, *to_match*)
Get all the rows that match a paricular element of a column. :param column: str :param to_match: str :rtype: pandas.DataFrame

get_row_matches(*column1*, *to_match*, *column2*)
Get the elements of the rows that match a particular column. If one element, this can be converted easily enough :param column1: str :param to_match: str :param column2: str :rtype: pandas.Series

n_matches(*column*, *to_match*)
Return the number of matches. :param column: str :param to_match: str :rtype: int

to_tsv(*path_or_buf=None*, *na_rep=''*, *float_format=None*, *columns=None*, *header=True*, *index=True*, *index_label=None*, *mode='w'*, *encoding=None*, *compression=None*, *quoting=None*, *quotechar='"'*, *line_terminator='\n'*, *chunksize=None*, *tupleize_cols=False*, *date_format=None*, *doublequote=True*, *escapechar=None*, *decimal='.'*)

jade.basic.pandas.PandasDataFrame.**detect_numeric**(*df*)
Detect numeric components

Parameters **df** – pd.DataFrame

Return type pd.DataFrame

jade.basic.pandas.PandasDataFrame.**drop_duplicate_columns**(*df*)
Drop Duplicate columns from the DataFrame. Return DF

Parameters **df** – pandas.DataFrame

Return type pandas.DataFrame

jade.basic.pandas.PandasDataFrame.**get_columns**(*df*, *columns*)
Get a new dataframe of only the columns

Parameters

- **df** – pandas.DataFrame
- **columns** – list

Return type pd.DataFrame

jade.basic.pandas.PandasDataFrame.**get_match_by_array**(*df*, *column*, *match_array*)
Get a new dataframe of all dataframes of the subset series, match_array

Note: This will result in a dataframe, but there may be strange issues when you go to plot the data in seaborn
No idea why.

Parameters

- **df** – pd.DataFrame
- **column** – str
- **match_array** – pd.Series

Return type pd.DataFrame

jade.basic.pandas.PandasDataFrame.**get_matches**(*df*, *column*, *to_match*)
Get all the rows that match a paricular element of a column.

Parameters

- **df** – pandas.DataFrame
- **column** – str
- **to_match** – str

Return type pd.DataFrame

jade.basic.pandas.PandasDataFrame.**get_multiple_matches**(*df*, *column*, *to_match_array*)
Get all the rows that match any of the values in to_match_array.

Parameters

- **df** – pandas.DataFrame
- **column** – str
- **to_match_array** – list

Return type pd.DataFrame

jade.basic.pandas.PandasDataFrame.**get_n_matches**(*df*, *column*, *to_match*)
Get the number of matches :param df: pd.DataFrame :param column: str :param to_match: :rtype: int

jade.basic.pandas.PandasDataFrame.**get_row_matches**(*df*, *column1*, *to_match*, *column2*)
Get the elements of the rows that match a particular column. If one element, this can be converted easily enough :param df: pd.DataFrame :param column1: str :param to_match: str :param column2: str :rtype: pd.Series

jade.basic.pandas.PandasDataFrame.**get_value**(*df*, *column*)
Get a single value from a one-row df. THis is to help for implicit docs, since the syntax to Iloc is so fucking strange.

Parameters

- **df** – pd.DataFrame
- **column** – str

Returns value

jade.basic.pandas.PandasDataFrame.**multi_tab_excel**(*df_list*, *sheet_list*, *file_name*)
Writes multiple dataframes as separate sheets in an output excel file.

If directory of output does not exist, it will create it.

Author: Tom Dobbs http://stackoverflow.com/questions/32957441/putting-many-python-pandas-dataframes-to-one-excel-worksh

Parameters

- **df_list** – [pd.Dataframe]
- **sheet_list** – [str]
- **file_name** – str

jade.basic.pandas.PandasDataFrame.**sort_on_list**(*df*, *column*, *sort_order*)
Given a list of values, and a column, create a new dataframe that is sorted like so. No idea why this is so difficult. :param df: :param list_to_sort: :rtype: pd.DataFrame

jade.basic.pandas.stats module

jade.basic.pandas.stats.**calculate_stddev**(*df*, *x*, *y*, *hue=None*)
Calcuates standard deviations for a normal distribution (Numerical data) over X and Hue categories.

If hue is given, the hue column will be added, and the overall will be of 'ALL'

Example DataFrame output (x='exp', y= 'length_recovery_freq', hue = 'cdr':

SD cdr exp y

20 6.739596 H2 ALL length_recovery_freq 21 7.373650 H2 min.remove_antigen-F length_recovery_freq 22 6.400637 ALL min.remove_antigen-T length_recovery_freq

Parameters

- **df** – pandas.DataFrame
- **x** – str
- **y** – str
- **total_column** – str
- **hue** – str

Return type pandas.DataFrame

jade.basic.pandas.stats.**calculate_stddev_binomial_distribution**(*df*, *x*, *y*, *total_column*, *y_mean_column*, *hue=None*)

Calculates standard deviations for a binomial distribution (like experiment True/False values) over X and Hue categories..

Typically used for bar-plot.

If hue is given the hue column will be added, and the overall will be of 'ALL', plus that of Hue

Example DataFrame output (x='exp', y= 'length_recovery_freq', hue = 'cdr':

SD cdr exp y

20 6.739596 H2 ALL length_recovery_freq 21 7.373650 H2 min.remove_antigen-F length_recovery_freq 22 6.400637 ALL min.remove_antigen-T length_recovery_freq

Parameters

- **df** – pandas.DataFrame
- **x** – str
- **y** – str
- **total_column** – str
- **hue** – str

Return type pandas.DataFrame

jade.basic.plotting package

jade.basic.plotting.MakeFigure module

class jade.basic.plotting.MakeFigure.**MakeFigure**(*rows=1*, *columns=1*, *share_x=True*, *share_y=True*)

Deprecated. NOW - GO Checkout SEABORN instead of this class! Essentially, this is an interface to a facet grid. Seaborn does this awesomely.

My take on a plotting interface. Because I think matplotlib's interface sucks.

I wrote this before I knew of pandas.

You need to know the number of plots ahead of time by passing the grid. 1x1 will make one plot. 2x2 will make a grid of 4 plots. 1x3 is 3 columns of grids horizontally 3x1 is a list of figures.

share_x and share_y tell the full sublplot to share the axis.

add_data(*x*, *y*, *label*)

add_grid(*x_grid=True*, *y_grid=True*)

fill_subplot(*title*, *labels*, *x_axis_label=None*, *y_axis_label=None*, *index=None*, *grid=None*, *add_legend=False*, *linestyle='–'*, *marker='^'*, *colors=None*)

This will add data to a particular subplot/plot.

: title: : labels: : x_axis_label: : y_axis_label: : specify_index: : add_legend: : linestyle: : marker: : colors: :return:

get_data(*label*)

get_labels()

get_plot(*n=0*)

Parameters **n** – int

Returns mpl.axes.SubplotBase

get_x_data(*label*)

get_y_as_list(*labels*)

get_y_data(*label*)

merge_data(*data_dict*, *replace_current_labels=False*)

reset(*rows=1*, *columns=1*, *share_x=True*, *share_y=True*)

save_plot(*outpath*, *tight=True*)

set_data(*data_dict*)

set_x_limits(*min*, *max*, *plot_num=None*)

set_x_scale(*scale='log'*, *plot_num=None*)

set_y_limits(*min*, *max*, *plot_num=None*)

set_y_scale(*scale='log'*, *plot_num=None*)

jade.basic.plotting.MakeFigure.**pad_single_title**(*ax*, *x=0.5*, *y=1.05*)

Move the Title up in reference to the plot, essentially adding padding. SINGLE AXES :param ax:Axes :param x: :param y: :return:

jade.basic.plotting.MakeFigure.**plot_general_pandas**(*df*, *title*, *outpath*, *plot_type*, *x*, *y=None*, *z=None*, *top_p=0.95*, *reverse=True*)

Plot anything in pandas. Make it look descent. Save the figure.

If you are doing this multiple times in a Notebook:

Don't forget to call (matplotlib.pyplot) plot.show() plot.close()

Parameters

- **df** – pandas.DataFrame
- **title** – str
- **outpath** – str
- **plot_type** – str
- **x** – str
- **y** – str
- **z** – str
- **top_p** – float
- **reverse** – bool

Return type matplotlib.Axes

jade.basic.plotting.MakeFigure.**plot_x_vs_y_sea_with_regression**(*df*, *title*, *outpath*, *x*, *y*, *top_p=0.95*, *reverse=True*)

Plot X vs Y using a Pandas Dataframe and Seaborn, with regression line., save the figure, and return the Axes.

If you are doing this multiple times in a Notebook:

Don't forget to call (matplotlib.pyplot) plot.show() plot.close()

Parameters

- **df** – pandas.DataFrame
- **title** – str
- **outpath** – str
- **x** – str
- **y** – str
- **top_p** – float
- **reverse** – bool

Return type matplotlib.Axes

jade.basic.plotting.MakeFigure.**set_common_title**(*fig*, *title*, *size=16*, *x=0*, *y=1.05*)
for FACETED plots, add a common title.

Parameters

- **fig** – Figure
- **title** – str
- **x** – int
- **y** – int

Returns

jade.basic.plotting.MakeFigure.**set_common_x_y_label**(*fig*, *x_text*, *y_text*)
For FACETED plots, add a common X or Y.

Parameters

- **fig** – Figure
- **x_text** – str
- **y_text** – str

Returns

jade.basic.plotting.correlations module

jade.basic.plotting.correlations.**annotate_r_value**(*data*, *x*, *y*, *ax*, *func=<function pearsonr>*, *template=None*, *stat=None*, *loc='best'*)
Forked from seaborn JointPlot for use with regplot, scatter, etc. Woot. Needs to actually go into Seaborn Now!

Annotate the plot with a statistic about the relationship.

data: pandas.DataFrame x: str y: str ax: matplotlib.Axes

func [callable] Statistical function that maps the x, y vectors either to (val, p) or to val.

template [string format template, optional] The template must have the format keys "stat" and "val"; if *func* returns a p value, it should also have the key "p".

stat [string, optional] Name to use for the statistic in the annotation, by default it uses the name of *func*.

loc [string or int, optional] Matplotlib legend location code; used to place the annotation.

jade.basic.plotting.error_bars module

jade.basic.plotting.error_bars.**calculate_set_errorbars_hist**(*ax*, *data*, *x*, *y*, *binomial_distro=True*, *total_column='total_entries'*, *y_freq_column=None*, *x_order=None*, *hue_order=None*, *hue=None*, *caps=True*, *color='k'*, *linewidth=0.75*, *base_columnwidth=0.8*, *full=True*)

Calculates the standard deviation of the data, sets erorr bars for a bar chart. Default base_columnwidth for seaborn plots is .8

Optionally give x_order and/or hue_order for the ordering of the columns. Make sure to pass this while plotting. Note:

If Hue is enabled, this base is divided by the number of hue_names for the final width used for plotting.

Parameters

- **ax** – mpl.Axes
- **data** – pandas.DataFrame
- **x** – str
- **y** – str
- **binomial_distro** – bool
- **total_column** – str
- **y_freq_column** – str
- **x_order** – list
- **hue_order** – list
- **hue** – str
- **caps** – bool
- **color** – str
- **linewidth** – float
- **base_columnwidth** – float
- **full** – bool

Return type None

jade.basic.plotting.error_bars.**calculate_set_errorbars_scatter**(*ax*, *data*, *x*, *y*, *binomial_distro=False*, *total_column='total_entries'*, *caps=False*, *color='k'*, *lw=1.5*)

(Untested) - Calculates the standard deviation of the data, sets error bars for a typical scatter plot

jade.basic.sequence package

jade.basic.sequence.ClustalRunner module

class jade.basic.sequence.ClustalRunner.**ClustalRunner**(*fasta_path*, *clustal_name='clustal_omega'*, *clustal_dir=None*)

A very simple class wrapper to run clustal omega.

output_alignment(*out_dir*, *out_name*, *parellel_process=False*)
Configure command line and Run Clustal Omega

set_extra_options(*extra_options=''*)
Set any extra options as a string which will be added to the end of the command line.

set_fasta_path(*fasta_path*)
Set the fasta path for alignment.

set_hard_wrap(*hard_wrap*)
Set the number of charactors before clustal will wrap. Usually 60-80.

set_output_format(*output_format*)
Set the output format

set_threads(*threads*)
Limit the number of threads for Clustal

jade.basic.sequence.PDBConsensusInfo module

class jade.basic.sequence.PDBConsensusInfo.**PDBConsensusInfo**(*resinfo_list*)

Class to compute frequency and probability from an array of PDBInfo classes. The sequences within PDBInfo do not necessarily need to be the same length. A given sequence position is identified and stored in the data maps by its [pdb_num, chain, and icode] -> Use get_position_from_residue(residue) to get this position from a Residue instance.

compute_stats()
Compute frequency and probability (0-1) for each position for each amino acid

get_all_sorted_positions()

get_consensus(*residue*)

get_consensus_for_position(*position*)

get_consensus_for_residues(*residue_list*)
Get the consensus for an ORDERED list of Residues

get_consensus_sequence()

get_frequency(*residue*, *aa*)

get_frequency_for_position(*position*, *aa*)

get_position_from_residue(*residue*)

get_probability(*residue*, *aa*)
Get probability of the current position (starting from 0) and aa

get_probability_for_position(*position*, *aa*)

init_data_map()
Sets all probabilities 0 and appends each map to the stats vector

output_seqlogo(*outdir*, *outname*, *clustalpath=None*)

output_seqlogo_bt_residues(*outdir*, *outname*, *res1*, *res2*, *chain*)

output_seqlogo_for_regions(*regions*, *outdir*, *outname*, *chain*)
Regions is an array of Regions classes. Basically start/stop points

return_initialized_total_map()

set_sequences(*pdb_info_list*)
Set a sequence list

jade.basic.sequence.SequenceInfo module

class jade.basic.sequence.SequenceInfo.**SequenceInfo**
Simple class for holding + accessing sequence metadata

Original class for sequence info. Basically deprecated by SequenceStats and PDBConsensusInfo.

get_chain()

get_end_residue()

get_length()

get_pdbID()

get_pdbpath()

get_region()

get_residue(*resnum*)
If region is given, resnum is residue number of PDB If not, resnum in Rosetta resnum

get_sequence()

get_start_residue()

set_pdbID(*pdbID*)

set_pdbpath(*pdbpath*)

set_region(*region*)

set_sequence(*sequence*)

jade.basic.sequence.SequenceResults module

class jade.basic.sequence.SequenceResults.**SequenceResults**

Simple class for holding, calculating, + accessing result data Residue Numbers are in Rosetta numbering.

Original class for sequence stats. Basically deprecated by SequenceStats and PDBConsensusInfo.

add_reference_residue(*resnum*, *one_letter_code*)

add_residue(*resnum*, *one_letter_code*, *decoy*)

get_all_mutated_positions()

get_all_reference_percent_observed()

Returns array of tripplets of [postion, one_letter_code, percent] of reference amino acid found.

get_all_residue_numbers()

get_all_residues_observed(*resnum*)

get_decoys_with_aa(*resnum*, *one_letter_code*)

Returns all decoys with a specific mutation at a position.

get_decoys_with_joint_aa(*resnum_one_letter_code_pair*)

Will output decoys that have x, y, z mutations at positions a, b, c

get_freq(*resnum*, *one_letter_code*)

get_percent(*resnum*, *one_letter_code*)

get_percent_string(*resnum*, *one_letter_code*)

get_reference_residue(*resnum*)

get_total(*resnum*)

jade.basic.sequence.SequenceStats module

class jade.basic.sequence.SequenceStats.**SequenceStats**(*sequence_list*)

Class for getting data from an array of strings of sequences (one letter code) of equal length.

compute_stats()

Compute frequency and probability (0-1) for each position for each amino acid

get_consensus_sequence()

get_frequency(*position*, *aa*)

get_probability(*position*, *aa*)

Get probability of the current position (starting from 0) and aa

init_data_map()

Sets all probabilities 0 and appends each map to the stats vector

return_initialized_total_map()

set_sequences(*sequence_list*)

Set a sequence list

jade.basic.sequence.fasta module

jade.basic.sequence.fasta.**chain_fasta_files_from_biostructure**(*structure*, *prefix*, *outdir*)

jade.basic.sequence.fasta.**chain_fasta_files_from_pose**(*pose*, *prefix*, *outdir*)
Creates fasta for each chain in the pose. Returns a list of paths for each fasta.

jade.basic.sequence.fasta.**chain_fasta_from_biostructure**(*structure*, *outname*, *outdir*)
Creates a single fasta from biopython structure, split by individual chains.

jade.basic.sequence.fasta.**chain_fasta_from_pose**(*pose*, *outname*, *outdir*)
Creates a single fasta from pose, split by individual chains.

jade.basic.sequence.fasta.**fasta_from_pose**(*pose*, *fasta_label*, *outname*, *outdir*)
Creates a fasta from the pose.

jade.basic.sequence.fasta.**fasta_from_sequences**(*sequences*, *outdir*, *outname*)
Output a general fasta, with tag being 1_outname etc. Use write_fasta for more control. Returns path to Fasta File written

jade.basic.sequence.fasta.**get_biochain_sequence**(*bio_chain*)

jade.basic.sequence.fasta.**get_label_from_fasta**(*fasta_path*)
Gets the first chainID found - Should be a single chain fasta file.

jade.basic.sequence.fasta.**get_sequence_from_fasta**(*fasta_path*, *label*)

jade.basic.sequence.fasta.**output_fasta_from_pdbs_biopython**(*path_header_dict*, *out_path*, *native_path=None*, *native_label='native'*, *is_camelid=False*)
Used only for L and H chains! Concatonates the L and H in order if present, otherwise assumes camelid at H.

jade.basic.sequence.fasta.**output_weblogo**(*alignment_path*, *outdir*, *outname*, *tag='Dunbrack Lab - Antibody Database Team'*)

jade.basic.sequence.fasta.**output_weblogo_for_sequences**(*sequences*, *outdir*, *outname*, *tag='Dunbrack Lab - Antibody Database Team'*)

jade.basic.sequence.fasta.**read_header_data_from_fasta**(*fasta_path*)
Reads > from fasta (PDBAA) and returns a defaultdict of pdb_chain: [method, residues, resolution, R factor]

jade.basic.sequence.fasta.**split_fasta_from_fasta**(*fasta_path*, *prefix*, *outdir*)
If we have a multiple fasta sequence, we split it into idividual files. Makes analysis easier. Returns list of paths for each fasta

jade.basic.sequence.fasta.**write_fasta**(*sequence*, *label*, *HANDLE*)
Writes a fasta with a sequence, chain, and open FILE handle. FULL Sequence on one line seems to be fine with HMMER3.

jade.basic.sql package

jade.basic.sql.StatementCreator module

class jade.basic.sql.StatementCreator.**StatementCreator**
Simple class for constructing a statement - allows on-the-fly addition of filters, cutoffs, etc.

add_FROM_string_or_strings(*strings_or_strings*)

add_ORDER_BY_string_or_strings(*string_or_strings*)

add_SELECT_string_or_strings(*string_or_strings*)

add_WHERE_string_or_strings(*string_or_strings*)

add_data_filter(*data_filter*)

create_statement()
Create the statment string from its components

jade.basic.structure package

jade.basic.structure.BasicPose module

class jade.basic.structure.BasicPose.**BasicPose**(*pdb_file_path=''*)

add_remark(*remark*)

change_occupancy()
Changes ALL occupancies in a PDB dictionary to 1.00 Returns PDB Dictionary.

clean_PDB()
Removes HSD, Waters: Tries to fix atom and residue name inconsistencies. HAS worked for changing a single MD pdb (NAMD) frame to Rosetta file. PLEASE Expand if possible to alias all residues for Rosetta compatability. NOT gaurenteed, but SHOULD work ok.

combine_pdb(*py_pdb*)
Combines pdb_map from instance of PyPDB to this one. Does not do any checks.

combine_pdb_map(*pdb_map*)
Combines pdb_map passed with the PythonPDBs map

copy_all_but_chains_into_pdb_map(*py_pdb*, *chains*)
Copies all data from one pdb_map of a py_pdb of all data except the specified chains into this one. Useful for reordering chains.

copy_chain_into_pdb_map(*py_pdb*, *chain*)
Copies all data from one pdb_map of a py_pdb of a chain into the one held in this class. Useful for reordering chains.

get_all_residues_of_type(*name3*)
Get PDB_Map subset of all residues of specific type

get_bb_data()
Get pdb_map subset of only N, CA, and C atoms

get_chain(*chain*)
Get Chain data as pdb_map subset

get_header()
Get 'header' of PDB as list of strings

get_hetatms()
Get hetatm data as pdb_map subset

get_pdb_map()

get_remarks()
Get 'REMARK' lines of PDB as a list of strings

get_residue(*resnum*, *chain*, *icode=''*)
Get PDB_Map subset of a specific residue

get_waters()
Get water data as pdb_map subset

morph_line_in_pdb_map_to_pdb_line(*entry*)
Oh What fun. ;) Magic Numbers?: (6,5,4,3,1,4,8,8,8,4,5);

pdb_alias(*pairs*, *element*)
Replaces ALL occurances of old element with new from pair. pair is a dictionary. In C++ it would be an array of pairs. [string old]:[string new] For Specific functions, please see below.

pdb_atom_alias(*line_num*, *pair*)
Replaces atom_names with ones Rosetta is happy with. pair is a dictionary. In C++ it would be an array of pairs. [string MD atom_name]:[string rosetta atom_name]

pdb_chain_alias(*pairs*)
Replaces ALL occurances of old chain with new chain. pair is a dictionary. In C++ it would be an array of pairs. [string old chain]:[string new chain]

pdb_residue_alias(*pairs*)
Replaces ALL occurances of old residue with new residue. pair is a dictionary. In C++ it would be an array of pairs. [string old residue_name]:[string new residue_name]

read_file_and_replace_b_factors(*deliminator*, *filename=''*, *resnum_column=1*, *chain_column=2*, *data_column=3*, *atomname_column=False*)
This function reads a deliminated file with data and inserts the data into the BFactor column. Used to visualize arbitrary data. Use function options to control which column the data is in as well as where your resnums and chains are located. If atomname column is given, will insert by atom instead of by residue

read_pdb_into_map()
Reads PDB file path into a basic PDB map. All data is held as strings.

remove_alternate_residues()
Removes any alternate residue codes and renumbers by renumbering from 1 and integrating any inserts.

remove_antigen()
Remove Antigen from an LH only PDB

remove_chain(*chain*)
Removes chain from pdb_map

remove_element_column()
Removes the extra stuff in the element column, but not the element itself.

remove_hetatm_atoms()

remove_residue_type(*name3*)

remove_waters()
Removes waters from pdb_map

replace_atom_b_factor(*resnum*, *chain*, *atomname*, *data*)
Replaces the b factor of an atom. Can be all string representations or not.

replace_residue_b_factor(*resnum*, *chain*, *data*)
Replaces the b factor of each atom in the residue with data. Can be all string representations or not.

save_PDB(*filename=False*, *output_remarks=True*, *output_header=True*)
Uses a the pdb_map to save the data as a PDB file.

set_pdb_map(*pdb_map*)

jade.basic.structure.BioPose module

class jade.basic.structure.BioPose.**BioPose**(*path*, *model_num=0*)
Bases: object

This is my biopython meta class. Because biopython's interface kinda sucks. This is a little cleaner.

The other way is to sublclass each Biopython class structure, which I'm not ready to do.

Right now, you need a path as I don't know how we would use this from sequence, etc as you do in Rosetta. :path: Is a path to an RCSB file. PDB (.pdb), mmCIF(.cif), and gzipped (.gz) versions.

atom(*atom_name*, *resnum*, *chain_id*, *icode=' '*, *alt=' '*, *model_num=0*)
Get a Bio Atom of the stored structure

Parameters

- **atom_name** – str
- **resnum** – int
- **chain_id** – str
- **icode** – str
- **alt** – str
- **model_num** – int

Return type bio.PDB.Atom.Atom

atoms(*resnum*, *chain_id*, *icode=' '*, *alt=' '*, *model_num=0*)
Get a list of Bio Atoms :param resnum: int :param chain_id: str :param icode: str :param alt: str :param model_num: int :rtype: list[bio.PDB.Atom.Atom]

chain(*chain_id*, *model_num=0*)
Get a Bio Chain of the stored structure :param chain_id: str :param model_num: int :rtype: bio.PDB.Chain.Chain

chains(*model_num=0*)
Get a list of Bio Chains :param model_num: int :rtype: list[bio.PDB.Chain.Chain

connected_to_next(*resi*, *peptide_bond_distance_cutoff=1.8*)

connected_to_previous(*resi*, *peptide_bond_distance_cutoff=1.8*)

get_chain_ids(*model_num*)
Get all chain IDS for a model. :param model_num: int :rtype: list[str]

get_chain_length(*chain_id*, *model_num=0*)
Get the number of AA in a chain - Not including alternate res locations :param chain_id: str :rtype: int

get_sequence(*chain_id*, *model_num=0*)
Get a sequence of a chain - Not including alternate res locations

Parameters

- **chain_id** – str
- **model_num** – int

Return type str

load_from_file(*path*)
Load a file from PDB or mmCIF. .gz is supported.

Parameters **path** – Path to PDB or mmCIF file

Return type tuple(bio.PDB.Structure.Structure, dict)

model(*model_num=0*)
Get a Bio Model of the stored structure :param id: int :rtype: bio.PDB.Model.Model

omega(*i*, *rosetta_definitions=True*)
Get the Omega Angle of i in radians Omega is defined as the dihedral angle between the peptide bond of i and i + 1, as in Rosetta. If rosetta_definitions are False, omega is then treated as being between i and i -1

Parameters

- **i** – int
- **reverse_rosetta_definitions** – bool

Return type float

pdbinfo()

phi(*i*)
Get the Phi Angle of i in radians

Parameters **i** – int

Return type float

psi(*i*)
Get the Psi Angle of i in radians

Parameters **i** – int

Return type float

reload_from_file(*path*, *model_num=0*)
Reload a BioPose from a file path. :param path: str :param model_num: int :return:

res_bond_distance(*resi*)
Get the stored bond distances between residue and residue+1 :param res: int :rtype: float

residue(*resnum*, *chain_id*, *icode=' '*, *alt=' '*, *model_num=0*)
Get a Bio Residue of the stored structure. Adds a chain_id attribute.

Parameters

- **resnum** – int
- **chain_id** – str
- **icode** – str
- **alt** – str
- **model_num** – int

Return type bio.PDB.Residue.Residue

residues(*chain_id*, *model_num=0*, *include_alt=False*)
Get residues, including or not including residues with alternate location codes - which can be a PITA Adds chain_id attribute to residue.

Parameters

- **chain_id** – str
- **model_num** – int
- **include_alt** – bool

Return type list[bio.PDB.Residue.Residue]

resnum(*pdb_num*, *chain*, *icode=' '*)

structure()
Get the Bio Structure stored in this class. :rtype: bio.PDB.Structure.Structure

total_residue()

jade.basic.structure.BioPose.**test_dihedrals**(*pose*)
Simple Test for Dihedral output

Parameters **pose** – BioPose

Return type bool

jade.basic.structure.BioPose.**test_pdbinfo**(*pose*)
Simple Test for pdbinfo output. :param pose: BioPose :rtype: bool

jade.basic.structure.SQLPose module

class jade.basic.structure.SQLPose.**PDB_database**(*database*)
This class is specifically for if we already have a database. Note: This is not a ROSETTA database. If you need to convert this, use ROSETTA (Which now works in PyRosetta!) Functions are to output the database as a PDB, output specific pieces of protein as a pdb and query the database.

query_all(*table='pdb'*)

query_chain(*table*, *chain*)

query_modelID(*table*, *modelID*)

query_pdbID(*table*, *pdbID*)

query_pdbID_and_chain(*table*, *pdbID*, *chain*)

query_pdbID_and_strucID(*table*, *pdbID*, *strucID*)

query_piece(*table*, *start*, *end*, *chain*)

query_piece_pdbID(*table*, *pdbID*, *start*, *end*, *chain*)

query_piece_pdbID_and_strucID(*table*, *pdbID*, *start*, *end*, *chain*, *strucID*)

query_strucID(*table*, *strucID*)

save_cur_as_pdb(*outpath*, *supress_modelSep=False*)
Saves the DB at the current cursor to a file. Make sure cursor is on the pdb table.

save_whole_db_as_db(*filename*, *seperate_structures=False*)
Saves the whole database in MEMORY to a file....

scrub(*table_name*)
This should help protect from sql injection. Not that it's important now, but... Author:OrangeOctopus from stack overflow

set_output_DIR(*outDIR*)

set_output_occupancy_1(*bool*)
Output structures with 1.0 as occupancy. Mainly for Rosetta use.

update_modelID_CDRS(*table*)
Updates modelID to specify L1 through H3 and framework for possible future statistical analysis.

class jade.basic.structure.SQLPose.**SQLPose**(*pdbID*, *modelID*, *structID*, *memory=False*, *path=''*)

fetch_and_read_pdb_into_database(*pdbID*, *read_header=False*, *header_only=False*)
Uses the PDB file specified, grabs it from the PDB, and reads the data in.

read_pdb_into_database_flat(*filePath*, *specific_chain=False*, *read_header=False*, *header_only=False*)
Reads the flat filepath specified into a database structure. This can then be parsed using the PDB_Database class. NOTE: Reading of header not implemented. if header_only is True, only loads the header. Useful for just getting specific information. More useful to D/L it from the pdb if possible. If Header only, reads the header into the database.

set_basic_options(*pdbID*, *modelID*, *structID*)

set_modelID(*modelID*)

set_pdbID(*pdbID*)

set_structID(*structID*)

jade.basic.structure.Structure module

class jade.basic.structure.Structure.**AntibodyResidueRecord**(*aa*, *pdb_res_num*, *chain*, *icode=' '*)
Bases: jade.basic.structure.Structure.ResidueRecord

Extension of Residue used to hold and access extra data used for renumbering/printing renumbering info I could backport python Enums, which would be incredibly useful here, but I don't want to require the additional step.

- used in Python3.4

get_cdr_type()

get_chain_type()
Gets the chaintype for the position - L or H

get_cluster()

get_distance()

get_gene()

get_meta()

get_old_chain()

get_old_icode()

get_old_resnum()

get_region()
Get the region type of the position - CDR/FRAMEWORK

is_cdr()

is_framework()

set_cdr_type(*cdr_type*)

set_chain_type(*chain_type*)

set_cluster(*cluster*)

set_distance(*distance*)

set_gene(*gene*)

set_meta(*meta*)

set_old_chain(*old_chain*)

set_old_icode(*old_icode*)

set_old_resnum(*old_resnum*, *icode*)
Sets the old resnum and icode in the numbering map dictionary.

set_region(*region*)

class jade.basic.structure.Structure.**AntibodyStructure**
Simple class for accessing Modified_AHO antibody numbering information outside of Rosetta.

get_cdr(*cdr_name*)
Get a CDR object of the given name :param cdr_name: str :rtype: CDR

get_cdr_names()
Get a list of cdr names :rtype: [str]

get_cdr_seq(*bio_pose*, *cdr_name*)
Get the CDR sequence from a bio pose :param bio_pose: basic.structure.BioPose :param cdr_name: str :rtype: str

get_cdrs()
Get a dictionary of CDR objects, indexed by name :rtype: defaultdict[str] = CDR

get_framework_info(*chain*)
Get the framework info class :param chain: str :rtype: FrameworkRegions

class jade.basic.structure.Structure.**Atom**(*name*)

class jade.basic.structure.Structure.**Bond**(*atom1*, *atom2*)

class jade.basic.structure.Structure.**CDR**(*name*)

add_residue_record(*name*, *num*)

get_pdb_chain()

get_pdb_end()

get_pdb_start()

set_gene(*gene*)

class jade.basic.structure.Structure.**FrameworkRegions**(*chain*)

F1()

F1_2()

F2_3()

F3()

get_regions()
Get the regions as an array of tupples

get_residue_record_region(*region_name*)

get_residue_record_regions()

get_start(*region*)

get_start_stop(*region*)

get_stop(*region*)

class jade.basic.structure.Structure.**PDBInfo**
Bases: object

Analogous to Rosetta PDBInfo Class I should start at 1

add_residue_record(*residue_record*)

clear()

delete_residue_record()
Delete the residue_record and renumber starting from 1

get_all_residue_records()
return all residue_records held as an array in order.

get_extra_data(*key*)

get_residue_record(*i*)
Get the residue record class for this particular index. :param i: :rtype: ResidueRecord

get_residue_record_of_pdb_num(*pdb_num*, *chain_id*, *icode=' '*)

get_resnum(*pdb_num*, *chain*, *icode=' '*)
Get the matching 'resnum' (i) or None if not found.

get_sequence()

get_sequence_bt_residue_records(*res1*, *res2*, *chain*)

get_sequence_bt_resnums(*start*, *stop*)

pdb2pose(*resnum*, *chain_id*, *icode=' '*)

pose2pdb(*i*)

res(*i*)

set_extra_data(*key*, *value*)

set_icode(*i*, *icode*)

set_pdb_num(*i*, *pdb_num*, *icode=' '*)

set_residue_record(*i*, *residue_record*)

total_residue()

total_residue_record()

total_residue_records()

class jade.basic.structure.Structure.**ResidueRecord**(*one_letter_aa*, *pdb_num*, *chain*, *icode=' '*)
Bases: object

Basic class to PDBInfo

get_aa()

get_chain()

get_extra_info(*key*)

get_extra_info_dict()

get_extra_info_keys()

get_icode()

get_pdb_num()

has_extra_info(*key*)

init_extra_infos(*array_of_keys*, *value*)

set_aa(*one_letter_aa*)

set_chain(*chain*)

set_extra_info(*key*, *value*)

set_icode(*icode*)

set_pdb_num(*pdb_num*)

tuple()

class jade.basic.structure.Structure.**ResidueRegion**(*res1*, *res2*, *name=None*)

get_name()

get_res1()

get_res2()

class jade.basic.structure.Structure.**electron**

class jade.basic.structure.Structure.**molecule**(*name*, *atomsormolecule*)

class jade.basic.structure.Structure.**nucleus**

class jade.basic.structure.Structure.**protein**

class jade.basic.structure.Structure.**protein_info**(*sequence*)

The protein has protein molecules. The PDB has a protein. Need to write this carefully.

attachDomains(*domain1*, *domain2*)

breakIntoDomains()

getInfo()

getPartners()

giveStructure(*pose*)

jade.basic.structure.util module

jade.basic.structure.util.**atomic_distance**(*res1*, *res2*, *res1_atom_name*, *res2_atom_name*)

Return the atomic distance between two arbitrary Bio residues and two arbitrary atom names. :param res1: Bio.PDB.Residue.Residue :param res2: Bio.PDB.Residue.Residue :param res1_atom_name: str :param res2_atom_name: str :rtype: float

jade.basic.structure.util.**get_biopython_structure**(*path*, *model_id=None*)

jade.basic.structure.util.**get_chain_length**(*bio_chain*)

jade.basic.structure.util.**get_num_biochains**(*model*)

jade.basic.structure.util.**get_seq_from_biochain**(*bio_chain*)

jade.basic.structure.util.**get_seq_from_biostructure**(*structure*, *chain_id*)

jade.basic.structure.util.**has_id**(*model*, *id*)
Returns true or false if the model has the chain. Because biopython is not updating it's index that has_id is using. WTF.

jade.basic.structure.util.**peptide_bond_distance**(*res1*, *res2*)
Return the bond distance between two residues using Numpy array math. :param res1: Bio.PDB.Residue.Residue :param res2: Bio.PDB.Residue.Residue :rtype: float

jade.basic.threading package

jade.basic.threading.JobDistributor module

class jade.basic.threading.JobDistributor.**JobDistributor**(*jobs*, *limit=10*, *verbose=True*)

execute()

set_jobs_limit(*limit*)

set_sleep_time(*time_*)

jade.basic.threading.Threader module

class jade.basic.threading.Threader.**Threader**(*print_interval=0*)
Bases: object

Class for starting 2 new threads. One that runs a system process and one that waits and prints info to std::out or whatever you currently have set as std::out. Use print interval to set the wait time between prints. Useful for GUI subprocessing.

run_functions(*functions*)
Run a bunch of lambda functions together with multiprocessing :param functions: :return:

run_system_command(*command*)
Run a system command using Popen. Prints out at end. Probably should remove this. :param command: :return:

class jade.basic.threading.Threader.**Threads**
Class for managing threads, killing them on program exit.

append(*thread*)

get_exitcode(*pid*)

is_alive(*pid*)

kill_all()

n_alive()

new_thread_allowed()

set_allowed_threads(*n*)

jade.basic.threading.Threader.**print_loop**(*p*, *print_interval=0*)

jade.basic.threading.Threader.**test_function**(*i*, *extra=''*)

jade.basic.RestypeDefinitions module

class jade.basic.RestypeDefinitions.**ResTypeSergey**(*ignore_groups=[]*)
Bases: object

Residue Types corresponding to Sergey Menis' definition of groups.

common_groups(*three_letter_one*, *three_letter_two*)
Return the intersection of groups. :return:

get_groups(*three_letter_code*)

has_common_group(*three_letter_one*, *three_letter_two*)

class jade.basic.RestypeDefinitions.**RestypeDefinitions**

get_all_one_letter_codes()

get_mutation_info()

get_one_letter_from_three(*three_letter_code*)

get_residue_info()

get_three_letter_from_one(*one_letter_code*)

is_conserved(*query_three_letter*, *group_three_letter*)

set_mutation_info()

set_residue_info()

jade.basic.general module

jade.basic.general.**extract_score_from_decoy**(*pdb_path*)
Extract total score from a rosetta decoy (gzipped or otherwise)

If score is not found, it will return 0.

Parameters **pdb_path** –

Returns float

jade.basic.general.**fix_input_args**()
Enables options to be passed to ArgumentParser with dashes, but not single charactor ones. Example:

–rosetta_args "-out:prefix test -out:path:all my/dir/"

Normally, this would fail if you had declared an -o option to the ArgumentParser. This happens because although the quotes are being parsed correctly, the system is looking or options using the starting '-' charactor. If you give a quote and then a space, you will recieve no error.

This code essentially checks for single dashes and puts a space in front of them. Note that this does not work with single charactor options you are hoping to pass with a quote. Because there is no way to grab the input string from the system and fix it myself, for these it will have to have a space after the quotes. This at least fixes the most common use cases (Mostly for use with Rosetta.).

jade.basic.general.**get_all_combos**(*list_of_lists*)
Get all the position-specific combos of a list of lists.

This is taken directly from Stack Overflow: http://stackoverflow.com/questions/798854/all-combinations-of-a-list-of-lists

Parameters **list_of_lists** – A list of lists we would like combos of.

Return type list[list]

jade.basic.general.**get_platform**()
Get OS of the particular platform the toolkit is being run on.

jade.basic.general.**get_rosetta_program**(*program*, *mpi=True*, *compiler='gcc'*)
Get the set program

jade.basic.general.**get_today**()

jade.basic.general.**match_patterns**(*search_string*, *patterns*)
Uses RE to match multiple patterns. Returns boolean of success

Parameters

- **search_string** – str
- **patterns** – [str]

Return type boolean

jade.basic.general.**merge_dicts**(**dict_args*)
Given any number of dicts, shallow copy and merge into a new dict, precedence goes to key value pairs in latter dicts. (Pre-Python 3.5) (http://stackoverflow.com/questions/38987/how-to-merge-two-python-dictionaries-in-a-single-expression)

jade.basic.general.**strip_left**(*s*, *pattern*)
Strips a string left (start) of string if found. Otherwise, returns the original string.

Parameters

- **s** – str
- **pattern** – str

Return type str

jade.basic.general.**strip_right**(*s*, *pattern*)
Strips a string right (end) of string if found. Otherwise, returns the original string.

Parameters

- **s** – str
- **pattern** – str

Return type str

jade.basic.numeric module

jade.basic.numeric.**distance**(*x1*, *y1*, *z1*, *x2*, *y2*, *z2*)
Get the distance between variables. :param x1: float :param y1: float :param z1: float :param x2: float :param y2: float :param z2: float :rtype: float

jade.basic.numeric.**distance_numpy**(*array1*, *array2*)
Get the distance between two points :param array1: numpy.Array :param array2: numpy.Array :rtype: float

jade.basic.numeric.**geometric_mean**(*data*)
Get the geometric mean of the data. Useful for numbers that go from 0 -> and are a type of enrichment of the data.

Parameters **data** – numpy.Array

Returns float

jade.basic.numeric.**get_n_s**(*num*)
Get a string for a float at .2f

jade.basic.numeric.**get_perc**(*freq*, *total*)
Get percent

jade.basic.numeric.**get_s_perc**(*freq*, *total*)
Get string of percent

jade.basic.numeric.**linear_rescale**(*min*, *max*, *value*)
Linearly rescale a value to 0 and 1 using the min and max values. :param min: :param max: :param value: :rtype: float

jade.basic.numeric.**wrapto360**(*angle*)
Wrap a value on -180, 180 to 360.

Parameters **degrees** – float

Returns float

jade.basic.path module

jade.basic.path.**get_Jade_root**()
Get the root path of Jade directory. :rtype: str

jade.basic.path.**get_all_pdb_paths**(*directory*, *ext='.pdb'*)

jade.basic.path.**get_all_pdbs**(*directory*, *ext='.pdb'*)

jade.basic.path.**get_bin_path**()
Get the path to the Jade apps directory :rtype: str

jade.basic.path.**get_database_path**()
Get the path to the Jade Database :rtype: str

jade.basic.path.**get_database_testing_path**()
Get the path to the database testing file. :return:

jade.basic.path.**get_decoy_extension**(*decoy*)
Return the extension of the decoy. .pdb, .pdb.gz, .cif, .cif.gz, etc. :param decoy: str :rtype: str

jade.basic.path.**get_decoy_name**(*decoy*)
Get the decoy name from path or name, whether .pdb, .pdb.gz or no extension. :param decoy: :rtype:str

jade.basic.path.**get_decoy_path**(*decoy*, *alternate_paths=None*)
Search .pdb, .pdb.gz, .cif, .cif.gz, .xml, .xml.gz In addition, Search alternative search paths. Return found path or NONE.

Parameters

- **decoy** –

- **alternate_paths** –

:rtype:str

jade.basic.path.**get_directories_recursively**(*inpath*)
Get a list of directories recursively in a path. Skips hidden directories. :param inpath: str :rtype: list

jade.basic.path.**get_file_paths**(*pattern*, *dir*, *ext='.pdb'*)
Get file paths matching the exact pattern and extension. :param pattern: :param dir: :param ext: :return:

jade.basic.path.**get_make_get_dirs**(*root*, *dirs*)
Recursively make dirs and return the final path :param root: :param dirs: :rtype: str

jade.basic.path.**get_matching_pdbs**(*directory*, *pattern*, *ext='.pdb'*)
Get pdbs in a directory matching a pattern. :param directory: :param pattern: :param ext: :return:

jade.basic.path.**get_nnk_database_path**()

jade.basic.path.**get_pdb_path**(*decoy*, *alternate_paths=None*)

jade.basic.path.**get_rosetta_features_json_path**()

jade.basic.path.**get_rosetta_features_root**()
Get the path to Rosetta features directory through set ROSETTA3_DB env variable. :rtype: str

jade.basic.path.**get_rosetta_features_run_script**()
Get the path to Rosetta features script dir through the set ROSETTA3_DB env variable. :rtype: str

jade.basic.path.**get_rosetta_flags_path**()

jade.basic.path.**get_rosetta_json_run_path**()

jade.basic.path.**get_testing_inputs_path**()
Get the path to testing inputs (PDBs,fasta,etc.) :rtype:str

jade.basic.path.**get_testing_path**()
Get the path to the Jade testing directory :rtype: str

jade.basic.path.**get_xml_scripts_path**()
Get the path to the Rosetta xml script directory. Useful for variable substitutions. :rtype: str

jade.basic.path.**make_dir_if_not_exists**(*dir*)

jade.basic.path.**open_file**(*file_path*, *mode='r'*)
Open a file which can be gzipped.

Parameters

- **file_path** –
- **mode** –

Returns

jade.basic.path.**parse_contents**(*file_path*)
Return a list of (stripped) content, skipping empty lines and comments. :param file: string :rtype: list

jade.clustering package

jade.clustering.CaliburRunner module

class jade.clustering.CaliburRunner.**CaliburWrapper**(*caliburPath=<type 'set'>*)

ret_centers()
return array of top 2 clusters, and a corresponding array of sizes.

ret_neighbors(*resultsPath*)

ret_random_num_neighbors()

ret_threshold()
Returns threshold used in analysis.

run_calibur(*PDBLIST_Path*, *chain=False*, *Nter=False*, *Cter=False*, *threshold=0*)
Nter, Cter are not residue numbering - pretty much it is rosetta numbering as far as I can tell... Need to do this better with option type thing..

save_neighbors(*resultsPath*, *center*)

save_random_num_neighbors()

jade.machine_learning package

jade.machine_learning.util module

jade.machine_learning.util.**plot_2d_decision_regions**(*X*, *y*, *classifier*, *resolution=0.02*)

jade.machine_learning.util.**plot_decision_regions**(*X*, *y*, *classifier*, *test_idx=None*, *resolution=0.02*)

jade.pymol_jade package

jade.pymol_jade.PyMolScriptWriter module

class jade.pymol_jade.PyMolScriptWriter.**PyMolScriptWriter**(*outdir*)
Class to help build PyMol scripts using arbitrary lists of PDBs.

Example for loading all top models into PyMol, aligning them to the native, and labeling them:

scripter = PyMolScriptWriter(outpath)

if native_path: scripter.add_load_pdb(native_path, "native_"+os.path.basename(native_path))

scripter.add_load_pdbs(pdb_path_list, load_as_list) scripter.add_align_all_to(scripter.get_final_names()[0]) scripter.add_show("cartoon") scripter.add_line("center") scripter.add_save_session(pse_path) scripter.write_script("load_align_top.pml") run_pymol_script(top_dir+"/"+"load_align_top.pml")

add_align_all(*sele1=''*, *sele2=''*, *limit_to_bb=True*, *pair_fit=False*)
Align all to the first model

add_align_all_to(*model*, *sele1=''*, *sele2=''*, *limit_to_bb=True*, *pair_fit=False*)
Align all to a particular model

add_align_to(*model1*, *model2*, *sele1=''*, *sele2=''*, *limit_to_bb=True*, *pair_fit=False*)
Align one model to another, optionally specifying a selection. Recommended to use superimpose instead

add_antibody_script()
Add running the color cdrs pymol script. Antibody must be in AHO numbering

add_center(*sele=None*)

add_color(*sele*, *color*)
Add color to a selection. sele: PyMol Selection color: Particular color.

See Also self.colors

add_group_object(*name*, *new_group_name*)
Group a single object to another. Useful for meta-groups.

add_group_objects(*names*, *new_group_name*)
Group a set of pre-loaded names to the new group.

add_hide(*vis_type*, *sele=''*)
Hide a representation. Optionally with a particular selection.

add_line(*line*)
Add an arbitrary line to the script

add_load_pdb(*pdb_path*, *load_as=None*, *group=None*)
Add line to load a PDB Path into PyMol Optionally load them as a particular name Will then set the final names PyMol uses to the object.

add_load_pdbs(*pdb_paths*, *load_as=None*, *group=None*)
Add lines to load the list of PDB paths into PyMol Optionally load them as a particular name Will then set the final names PyMol uses to the object.

add_save_session(*session_path*)
Add a line to save the session to a FULL path

add_select(*name*, *sele*, *group=None*)

add_show(*vis_type*, *sele=''*)
Show a representation. Optionally with a particular selection

add_superimpose(*sele1*, *sele2*)
Super impose two selections using the super command

add_superimpose_all_to(*model*, *sele1*, *sele2*)

clear()

get_color_types()

get_colors_of_type(*color_type*)

get_final_names()
Get the final names PyMOL will use after loading PDBs.

get_sele(*chain*, *resid_array*)
Get a a selection from an array of residue IDs and a particular chain. If the residue Id is a two-element tupple, then add a selection between the first and last element

get_vis_types()

print_script()

reset_script()

run_script(*script_outname='pml_script.pml'*, *delete_script=True*, *parellel_process=False*)
Save and Run the Pymol script :param script_outname: str

save_script(*fname=None*)

set_outdir(*outdir*)

write_script(*fname=None*)

jade.pymol_jade.PyMolScriptWriter.**make_pymol_session_on_top**(*pdb_path_list*, *load_as_list*, *script_dir*, *session_dir*, *out_name*, *top_num=None*, *native_path=None*, *antibody=True*)

Make a pymol session on a set of decoys. Usually an ordered decoy list. :param top_dir: :param pdb_path_list: List of PDB Paths :param load_as_list: List of PDB Path names for pymol. :param outdir: :param out_name: :param top_num: :param native_path: :return:

jade.pymol_jade.PyMolScriptWriter.**make_pymol_session_on_top_ab_include_native_cdrs**(*pdb_path_l*
load_as_lis
script_dir,
ses-
sion_dir,
out_name,
cdr_dir,
top_num=1
na-
tive_path=

Make a pymol session on a set of decoys. These decoys should have REMARK CDR_origin. These origin pdbs will be aligned and included in the pymol session :param top_dir: :param pdb_path_list: List of PDB Paths :param load_as_list: List of PDB Path names for pymol. :param cdr_dir: The directory of antibody CDRs from PyIgClassify. :return:

jade.pymol_jade.PyMolScriptWriter.**make_pymol_session_on_top_scored**(*pdbpaths_scores*, *script_dir*, *session_dir*, *out_name*, *top_num=-1*, *native_path=None*, *antibody=True*, *parellel=True*, *super=''*, *run_pymol=True*, *model_names=[]*)

Make a pymol session on a set of decoys with a tuple of [[score, pdb], ...] Optionally, it can be a 3 length tupple with model name to use as last:

[[score, pdb, model_name], ...]

if run_pymol is False, will not run pymol.

Pymol names will be: model_n_RosettaModelNumber_score Score will be truncated to two decimal places.

Returns configured PyMol Scripter for extra use.

Parameters

- **pdbpaths_scores** – tuple of [[score, pdb], ...]
- **script_dir** – Path to output PyMol script
- **session_dir** – Path to output Session

- **out_name** – name of the Pymol session
- **top_num** – Optional - Only output TOP N models
- **native_path** – Optional - Path to any input native to add to pymol session
- **parellel** – Optional - Run in parellel (so many pymol sessions can be created at once)
- **super** – Optional - Super to THIS particular selection instead of align_all to.
- **run_pymol** – Optional - Run Pymol using script? Default true

Return type PyMolScriptWriter

jade.pymol_jade.PyMolScriptWriter.**run_pymol_script**(*script_path*, *run_gui=False*, *delete_script=False*, *parellel_process=True*)

Run the script of the given path.

jade.RAbD package

Subpackages

jade.RAbD.window_main package

jade.RAbD.window_main.AnalysisFrame module

class jade.RAbD.window_main.AnalysisFrame.**AnalysisFrame**(*main*, *compare_designs*, *main_gui*, ***options*)

Bases: Tkinter.Frame

check_set_pyigclassify()

copy_to_dir()

copy_to_dir_and_rename()

get_decoys()

Split the decoys by return type - so you can do multiple things to the decoys :rtype: list of str

open_msa()

open_seq_logo()

print_decoy_info()

print_enrichments()

print_fasta()

print_recovery()

set_tk()

sho_tk()

jade.RAbD.window_main.CompareStrategiesFrame module

class jade.RAbD.window_main.CompareStrategiesFrame.**CompareStrategiesFrame**(*main*, *compare_designs*, *main_gui*, ***options*)

Bases: Tkinter.Frame

add_main_strategy()

add_to_current(*from_listbox*, *to_listbox*)

delete_current(*listbox*)

get_full_strategy_list()

populate_all_strategies()

set_tk()

sho_tk()

show_strat_items()

jade.RAbD.window_main.FeaturesFrame module

class jade.RAbD.window_main.FeaturesFrame.**FeaturesFrame**(*main*, *compare_designs*, *main_gui*, ***options*)

Bases: Tkinter.Frame

run_features_reporter(*type*)

set_tk()

sho_tk()

jade.RAbD.window_main.menu module

class jade.RAbD.window_main.menu.**AntibodyDesignAnalysisMenu**(*main*, *compare_designs*, *main_gui*)

Bases: object

camelid_tracer(*name*, *index*, *mode*)

change_root()

create_subset_databases(*score_name*)

get_full_strategy_list()

open_sequence_logo()

print_threads()

read_from_db_dir_set_strategies()

run_clustal_omega()

run_clustal_on_all_combined()

run_copy_all()

set_clustal_output_format()

set_clustal_soft_wrap()

set_max_clustal_procs()

set_native_path()

set_pyigclassify_dir()

set_reference_db()

set_tk()

set_top_n()

set_top_n_combined()

sho_tk()

jade.RAbD.AnalyzeAntibodyDesigns module

class jade.RAbD.AnalyzeAntibodyDesigns.**CompareAntibodyDesignStrategies**(*db_dir*, *out_dir_name*, *strategies=[]*, *jsons=[]*)

Class mainly for comparing different Antibody Design strategies using our Features Databases.

copy_all_models()

copy_top()

create_score_subset_database(*score_name*, *prefix*, *features_type='antibody'*)

get_csv_data(*top=False*, *summary=False*)

Get data by converting everything to a pandas dataframe first. For now, one function pretty much does everything.

Return type [pandas.Dataframe],[str]

get_db_path(*strategy*, *features_type='antibody'*)

get_full_features_type(*type*)

get_pandas_dataframe()

Gets a pandas Dataframe for all :rtype: pandas.DataFrame

get_strategies()

get_top_dataframe_by_all_scores()

Get a pandas DataFrame for top, grouped by the type of score that is on. :rtype: pandas.DataFrame

get_top_from_dataframe(*score_name*)

Gets a pandas Dataframe for top :rtype: pandas.DataFrame

output_all_data_as_excel_file(*top=True*)

output_csv_data(*top=False*, *summary=False*)

Output a CSV file of combined or individual data.

output_len_or_clus_alignment(*alignment_type*, *features_type='antibody'*)

output_len_or_clus_enrichment(*alignment_type*, *features_type='antibody'*)

output_len_or_clus_recovery(*alignment_type*, *features_type='antibody'*)

output_stats()
Depracated in favor of dataframe summaries.

run_clustal_omega(*processors*, *output_format='fasta'*, *extra_options=''*)

run_clustal_omega_on_all_combined(*processors*, *output_format*, *extra_options=''*)

run_features(*type*, *plot_name=''*)

set_cdrs_from_list(*cdr_list*)

set_strategies(*strategies*)

set_strategies_from_databases()
Set the strategies from the db_dir/databases directory :return:

set_strategies_from_db_dir_top_dir()

set_strategies_from_json_infos()
Uses self.json, which are AnalysisInfo classes, to populate.

Returns

class jade.RAbD.AnalyzeAntibodyDesigns.**Perc**(*count*, *total*)
Simple class for holding enrichment/recovery information

get_count()

get_formated_perc(*perc*)

get_perc_decimal()

get_perc_whole()

get_total()

jade.RAbD.AnalyzeAntibodyDesigns.**calculate_enrichments**(*all_decoy_data*, *cdr*, *decoy_list=None*)
Returns defaultdict of [count_type] : Perc

jade.RAbD.AnalyzeAntibodyDesigns.**calculate_observed_value**(*value*, *all_decoy_data*, *cdr*, *decoy_list=None*)
Calculate the enrichment of some value

jade.RAbD.AnalyzeAntibodyDesigns.**calculate_recovery**(*native_data*, *all_decoy_data*, *cdr*, *decoy_list=None*)
Calculate the recovery of some value to native Returns

jade.RAbD.AnalyzeAntibodyDesigns.**count_native_matches**(*decoy_data*, *native_data*, *cdrs*)

jade.RAbD.AnalyzeAntibodyDesigns.**get_star_if_native**(*decoy_data*, *native_data*, *cdr*)

jade.RAbD.AnalyzeAntibodyDesigns.**get_str**(*value*)

jade.RAbD_BM package

jade.RAbD_BM.AnalysisInfo module

class jade.RAbD_BM.AnalysisInfo.**AnalysisInfo**(*json_path*)
Simple class that parses a json file which defines (USING RELATIVE PATHS):

1. exp - The name of the experiment - whatever you want it to be.

2. decoy_dir - the directory of the decoys.
3. features_db - the db where the features reporters have been run.

The class will store this information, and parse the benchmark info in the decoy dir, storing a BenchmarkInfo object. Benchmark classes and scripts will take lists to these analysis files and use them to generate plots and data.

get_bm_info()
Get the benchmark info :rtype: rosetta_bms.BenchmarkInfo

get_decoy_dir()

get_exp()

get_features_db()

class jade.RAbD_BM.AnalysisInfo.**NativeInfo**(*dataset*, *input_pdb_type*, *root_dataset_dir='datasets'*)
Simple class to hold native information.

get_decoy_dir()

get_features_db()

jade.RAbD_BM.AnalyzeRecovery module

class jade.RAbD_BM.AnalyzeRecovery.**AnalyzeRecovery**(*pyig_design_db_path*, *analysis_info*, *native_info*, *cdrs=None*)
Pools Recovery and RR data, outputs to DB

apply(*db_path*, *drop_tables=False*)
Calculate and Output all the data to the given database.

Parameters **db_path** – str

initialize()
Initialize ALL input data before calculating and outputing everything.

class jade.RAbD_BM.AnalyzeRecovery.**ObservedRecoveryCalculator**(*native_db_path*)
Bases: jade.RAbD_BM.AnalyzeRecovery.RecoveryCalculator

apply(*exp_name*, *pdbids*, *cdrs*, *bm_decoy_path*, *output_dir='data'*)
Calculates the number of times the native clusters and lengths were observed during the experiment, for each PDB. Returns the resulting dataframe.

Return type pandas.DataFrame

class jade.RAbD_BM.AnalyzeRecovery.**PyIgClassifyDBRepresentationCalculator**(*native_db_path*)
Bases: jade.RAbD_BM.AnalyzeRecovery.RecoveryCalculator

apply(*exp_name*, *cdrs*, *pyig_db_path*, *lambda_kappa_dict*, *output_dir='data'*)
Calculates the number of times lengths and clusters are present in the PyIgClassify database. :param lambda_kappa_dict : dict-like ['lambda'] = [pdbid,]

Return type pandas.DataFrame

class jade.RAbD_BM.AnalyzeRecovery.**RecoveryCalculator**(*native_db_path*)
Bases: object

class jade.RAbD_BM.AnalyzeRecovery.**TopRecoveryCalculator**(*native_db_path*)
Bases: jade.RAbD_BM.AnalyzeRecovery.RecoveryCalculator

apply(*exp_name*, *pdbids*, *cdrs*, *bm_db_path*, *output_dir='data'*)
Calculate length and cluster recoveries. Store them the same way we used to for the recovery parser. Returns the resulting dataframe of recoveries. :rtype: pandas.DataFrame

jade.RAbD_BM.AnalyzeRecovery.**calculate_exp_rr_and_recovery**(*exp*, *result_df*)
Calculate the overall recovery and risk ratio. :param exp: :param result_df: :rtype: pandas.DataFrame

jade.RAbD_BM.AnalyzeRecovery.**calculate_per_cdr_rr_and_recovery**(*exp*, *cdrs*, *result_df*)
Calculate the recovery and risk-ratios PER CDR. :rtype: pandas.DataFrame

jade.RAbD_BM.AnalyzeRecovery.**calculate_recovery_and_risk_ratios**(*top_recovery_df*, *observed_df*)
Calculate the Risk Ratio and Recovery Percent for each pdb/cdr given dataframes output by the calculators below.

Return a merged dataframe of the top recovery and observed, with the resulting risk ratio data.

Parameters

- **top_recovery_df** – pandas.DataFrame
- **observed_df** – pandas.DataFrame

Return type pandas.DataFrame

jade.RAbD_BM.AnalyzeRecovery.**get_decoys**(*input_dir*, *pdbid*)
Use GLOB to Match on pdbid for file names in the input dir. This should skip all the extra PDBs like excn, initial, relax, etc. :param input_dir: str :param tag: str

jade.RAbD_BM.RunBenchmarksRAbD module

class jade.RAbD_BM.RunBenchmarksRAbD.**RunBenchmarksRAbD**
Bases: jade.rosetta_jade.RunRosettaBenchmarks.RunRosettaBenchmarks

Benchmark class specifically for RAbD

Details:

ALL INPUT PDBs should go into

project_root/datasets

Typically, you will have multiple directories - native, relaxed, etc.

This is specified as a benchmark using 'input_pdb_type' in your json file.

ALL PDBLISTs for benchmarking should go into

project_root/datasets/pdblists

run_benchmark(*benchmark_names*, *benchmark_options*)
Run a single benchmark with options.

Parameters

- **benchmark_names** – List of benchmark names
- **benchmark_options** – List of benchmark options

Returns

jade.RAbD_BM.benchmark_plotting module

class jade.RAbD_BM.benchmark_plotting.**NativeCDRData**(*datatype*, *native_path*, *data_table='cdr_metrics'*)

get_all_data()

get_data(*pdbid*, *cdr*)

setup_data(*datatype*)

class jade.RAbD_BM.benchmark_plotting.**PlotData**(*native_data*, *rec_data*)

get_xy_of_exp(*exp*, *rec=True*, *skip_H3=True*)

plot_data(*outname*, *rec=True*)

class jade.RAbD_BM.benchmark_plotting.**RecoveryCDRData**(*db_paths*, *type='length'*)

setup_data()

jade.RAbD_BM.recovery_rr_tools module

jade.RAbD_BM.recovery_rr_tools.**calculate_geometric_means_rr**(*df*, *x*, *y*, *hue=None*)
Example use: rr_data_lengths = calculate_geometric_means_rr(df_all, x='cdr', y='length_rr', hue='exp') rr_data_clusters = calculate_geometric_means_rr(df_all, x='cdr', y='cluster_rr', hue='exp')

jade.RAbD_BM.recovery_rr_tools.**calculate_rr_errors**(*df_all_errors*)
Calculates the risk ratio errors for cluster and lengths using propagation error equations calculated for the recovery itself. Which is the same for percent as it would be raw data, as the N cancels out in the equations. http://lectureonline.cl.msu.edu/~mmp/labs/error/e2.htm

jade.RAbD_BM.recovery_rr_tools.**calculate_set_errorbars_hist**(*ax*, *data*, *x*, *y*, *binomial_distro=True*, *total_column='total_entries'*, *y_freq_column=None*, *x_order=None*, *hue_order=None*, *hue=None*, *caps=False*, *color='k'*, *linewidth=0.75*, *base_columnwidth=0.8*, *full=True*)
Calculates the standard deviation of the data, sets erorr bars for a histogram. Default base_columnwidth for seaborn plots is .8

Optionally give x_order and/or hue_order for the ordering of the columns. Make sure to pass this while plotting.

Notes:

1. If Hue is enabled, this base is divided by the number of hue_names for the final width used for plotting.
2. Caps are the line horizontal lines in the errorbar.
3. 'full' means error bars on both vertical sides of the histogram bar.

Warning: linewidth of .5 does not show up in all PDFs for all bars.

jade.RAbD_BM.recovery_rr_tools.**calculate_set_errorbars_scatter**(*ax*, *data*, *x*, *y*, *binomial_distro=False*, *total_column='total_entries'*, *caps=False*, *color='k'*, *lw=1.5*)

(Untested) - Calculates the standard deviation of the data, sets error bars for a typical scatter plot

jade.RAbD_BM.recovery_rr_tools.**calculate_stddev_binomial_distribution2**(*df*, *x*, *y*, *total_column*, *y_mean_column*, *hue=None*, *percent=True*)

Calcuates stddeviations for a binomial distribution. Returns a dataframe of stddevs If percent=True, we dived by the total to normalize the standard deviation. SD of 'mean' = SQRT(n*p*q) where p is probability of success and q is probability of failure.

jade.RAbD_BM.recovery_rr_tools.**load_precomputed_recoveries**(*db_path='data/all_recovery_and_risk_ratio_d*
table='full_data')

Reads recovery data from a database created via script.

rtype: pandas.Dataframe

jade.RAbD_BM.recovery_rr_tools.**order_by_row_group**(*df*, *column*, *groups*)

Order a dataframe by groups. Return the dataframe. Probably a better way to do this already, but I don't know what it is.

jade.RAbD_BM.recovery_rr_tools.**plot_rr**(*data*, *x*, *y*, *hue=None*, *ci=None*)

jade.RAbD_BM.recovery_rr_tools.**remove_pdb_and_cdr**(*df*, *pdbid*, *cdr*)

Removes a particular pdbid and cdr from the db. Returns the new df.

jade.RAbD_BM.recovery_rr_tools.**set_errorbars_bar**(*ax*, *data*, *x*, *y*, *error_dfs*, *x_order=None*, *hue_order=None*, *hue=None*, *caps=False*, *color='k'*, *linewidth=0.75*, *base_columnwidth=0.8*, *full=True*)

Sets erorr bars for a bar chart.

Default base_columnwidth for seaborn plots is .8

Optionally give x_order and/or hue_order for the ordering of the columns. Make sure to pass this while plotting.

Notes:

1. If Hue is enabled, this base is divided by the number of hue_names for the final width used for plotting.
2. Caps are the line horizontal lines in the errorbar.
3. 'full' means error bars on both vertical sides of the histogram bar.

Warning: linewidth of .5 does not show up in all PDFs for all bars.

jade.RAbD_BM.recovery_rr_tools.**set_errorbars_bar_rr**(*ax*, *data*, *x*, *y*, *error_dfs*, *x_order=None*, *hue_order=None*, *hue=None*, *caps=False*, *color='k'*, *linewidth=0.75*, *base_columnwidth=0.8*, *full=True*)

Sets erorr bars for a bar chart.

Default base_columnwidth for seaborn plots is .8

Optionally give x_order and/or hue_order for the ordering of the columns. Make sure to pass this while plotting.

Notes:

1. If Hue is enabled, this base is divided by the number of hue_names for the final width used for plotting.
2. Caps are the line horizontal lines in the errorbar.
3. 'full' means error bars on both vertical sides of the histogram bar.

Warning: linewidth of .5 does not show up in all PDFs for all bars.

jade.RAbD_BM.tools module

jade.RAbD_BM.tools.**get_lambda_kappa_pdb_ids**(*dataset*, *pdb_type*, *root_dataset_dir='datasets/pdblists'*)

Get two lists: lambda and kappa pdbids

Parameters

- **dataset** – str
- **root_dataset_dir** – str

Return type [str],[str]

jade.RAbD_BM.tools.**get_pdb_paths**(*in_dir*, *exp_name*, *match_name=None*, *use_ensemble=False*)

jade.RAbD_BM.tools_ab_db module

jade.RAbD_BM.tools_ab_db.**get_all_clusters_for_length**(*db*, *cdr*, *length*, *limit_to_known=True*, *res_cutoff=2.8*, *rfac_cutoff=0.3*)

Get all unique clusters for a length and a cdr

jade.RAbD_BM.tools_ab_db.**get_all_lengths**(*db*, *cdr*, *limit_to_known=True*, *res_cutoff=2.8*, *rfac_cutoff=0.3*)

Get all unique lengths for a CDR

jade.RAbD_BM.tools_ab_db.**get_cdr_data_table_df**(*db_path*)

Get a dataframe with typical info from the cdr_data table in the PyIgClassify db. :param db_con: sqlite3.con :rtype: pandas.DataFrame

jade.RAbD_BM.tools_ab_db.**get_cdr_rmsd_for_entry**(*db*, *pdb*, *original_chain*, *cdr*, *length*, *fullcluster*)

jade.RAbD_BM.tools_ab_db.**get_center_dih_degrees_for_cluster_and_length**(*db*, *cdr*, *length*, *cluster*)
Returns a dictionary of center dihedral angles in positional order. Or returns False if not found. result["phis'] = [phis as floats] result["psis"] = [Psis as floats] result["omegas"] = [Omegas as floats]

jade.RAbD_BM.tools_ab_db.**get_center_for_cluster_and_length**(*db*, *cdr*, *length*, *cluster*, *data_names_array*)

jade.RAbD_BM.tools_ab_db.**get_cluster_enrichment**(*df*, *gene*, *cdr*, *cluster*)
Get the number of matches in the df and pdbid to the cdr and cluster :param df: pandas.DataFrame :rtype: int

jade.RAbD_BM.tools_ab_db.**get_cluster_matches**(*df*, *gene*, *cdr*, *cluster*)
Get a dataframe of the matching ("Recovered") rows (DataFrame).

Parameters **df** – pandas.DataFrame

Return type pandas.DataFrame:

jade.RAbD_BM.tools_ab_db.**get_data_for_cluster_and_length**(*db*, *cdr*, *length*, *cluster*, *data_names_array*, *limit_to_known=True*, *res_cutoff=2.8*, *rfac_cutoff=0.3*)
Get a set of data of a particular length, cdr, and cluster. data_names_array is a list of the types of data. Can include DISTINCT keyword

Example: data_names_array = ["PDB", "original_chain", "new_chain", "sequence"]

jade.RAbD_BM.tools_ab_db.**get_dihedral_string_for_centers**(*db*, *limit_to_known=True*)

jade.RAbD_BM.tools_ab_db.**get_length_enrichment**(*df*, *gene*, *cdr*, *length*)
Get the number of matches in the df and pdbid to the cdr and length

Parameters

- **df** – pandas.DataFrame
- **length** – int

Return type int

jade.RAbD_BM.tools_ab_db.**get_length_matches**(*df*, *gene*, *cdr*, *length*)
Get a dataframe of the matching ("Recovered") rows (DataFrame).

Parameters

- **df** – pandas.DataFrame
- **length** – int

Return type pandas.DataFrame

jade.RAbD_BM.tools_ab_db.**get_pdb_chain_subset**(*db*, *gene*)
Return a list of tuples of [pdb, chain] of the particular gene

jade.RAbD_BM.tools_ab_db.**get_stem_rmsd_for_entry**(*db*, *pdb*, *original_chain*, *cdr*, *length*, *fullcluster*)

jade.RAbD_BM.tools_ab_db.**get_total_entries**(*df*, *gene*, *cdr*)
Get a the total number of entries matching the gene and the cdr. Used for recovery. :param df: pandas.DataFrame :rtype: int

jade.RAbD_BM.tools_ab_db.**get_unique_sequences_for_cluster**(*db*, *cluster*, *include_outliers*, *outlier_definition='conservative'*)

jade.RAbD_BM.tools_features_db module

jade.RAbD_BM.tools_features_db.**get_all_entries**(*df*, *pdbid*, *cdr*)
Get all entries of a given PDBID and CDR. :param df: pandas.DataFrame :rtype: pandas.DataFrame

jade.RAbD_BM.tools_features_db.**get_cdr_cluster_df**(*db_path*)
Get a dataframe with typical cluster info in it, which was generated by the features reporter framework. :param db_con: sqlite3.con :rtype: pandas.DataFrame

jade.RAbD_BM.tools_features_db.**get_cluster**(*df*, *pdbid*, *cdr*)
Get the fullcluster from the dataframe for native or experimental data

Parameters **df** – pandas.DataFrame

Return type str

jade.RAbD_BM.tools_features_db.**get_cluster_matches**(*df*, *pdbid*, *cdr*, *cluster*)
Get a dataframe of the matching ("Recovered") rows (DataFrame).

Parameters **df** – pandas.DataFrame

Return type pandas.DataFrame:

jade.RAbD_BM.tools_features_db.**get_cluster_recovery**(*df*, *pdbid*, *cdr*, *cluster*)
Get the number of matches in the df and pdbid to the cdr and cluster :param df: pandas.DataFrame :rtype: int

jade.RAbD_BM.tools_features_db.**get_length**(*df*, *pdbid*, *cdr*)
Get the length from the dataframe for native or experimental data

Parameters **df** – pandas.DataFrame

Return type int

jade.RAbD_BM.tools_features_db.**get_length_matches**(*df*, *pdbid*, *cdr*, *length*)
Get a dataframe of the matching ("Recovered") rows (DataFrame).

Parameters

- **df** – pandas.DataFrame
- **length** – int

Return type pandas.DataFrame

jade.RAbD_BM.tools_features_db.**get_length_recovery**(*df*, *pdbid*, *cdr*, *length*)
Get the number of matches in the df and pdbid to the cdr and length

Parameters

- **df** – pandas.DataFrame
- **length** – int

Return type int

jade.RAbD_BM.tools_features_db.**get_total_entries**(*df*, *pdbid*, *cdr*)
Get the total number of entries of the particular CDR and PDBID in the database :param df: pandas.DataFrame :rtype: int

jade.rosetta_jade package

jade.rosetta_jade.BenchmarkInfo module

class jade.rosetta_jade.BenchmarkInfo.**BenchmarkInfo**(*decoy_path*, *full_name*, *final_name*, *scorefunction='talaris2014'*)
Simple Class for holding info for a particular benchmark. Parses the Run_Settings.txt file in the decoy directory. This file is output by RunRosettaBenchmarks.

The settings dictionary then holds key/value pairs. Here is an example of this file for RAbD:

CDR = ALL DATASET = bm2_ten DOCK = False INNER_CYCLE_ROUNDS = 1 INPUT_PDB_TYPE = pareto L_CHAIN = kappa MINTYPE = relax OUTER_CYCLE_ROUNDS = 100 PAPER_AB_DB = True PROTOCOL = even_cluster_mc RANDOM_START = True REMOVE_ANTIGEN = True SEPARATE_CDRS = False

get_dataset()
Get the dataset used for benchmarking. :rtype: str

get_decoy_path()
Get the directory of all of the decoys for this benchmark. :rtype:

get_final_name()
Get the final name of the benchmark (used mainly for features dbs or comparisons between benchmarks.) :rtype: str

get_full_name()
Get the full name of the benchmark. :rtype: str

get_input_pdb_type()
Get the input pdb type used, ex: native vs pareto :rtype: str

get_scorefunction_name()
Get the scorefunction name set in this info instance. :rtype: str

has_log_path()

jade.rosetta_jade.BenchmarkInfo.**get_run_settings**(*dir*, *fname='RUN_SETTINGS.txt'*)
Gets a dict of the settings used to run the benchmark in the directory.

The settings file looks like this, and is output by RunRosettaBenchmarks into the decoy directory:

CDR = ALL DATASET = bm2_ten DOCK = False INNER_CYCLE_ROUNDS = 1 INPUT_PDB_TYPE = pareto L_CHAIN = kappa MINTYPE = relax OUTER_CYCLE_ROUNDS = 100 PAPER_AB_DB = True PROTOCOL = even_cluster_mc RANDOM_START = True REMOVE_ANTIGEN = True SEPARATE_CDRS = False

Parameters **dir** – str

Return type defaultdict

jade.rosetta_jade.FeaturesJsonCreator module

class jade.rosetta_jade.FeaturesJsonCreator.**JsonCreator**(*out_path*, *script_type*)
Basic implementation of a simple JsonCreator to create Jsons. Could be expanded to not load jsons with pre-set scripts. A nicer implementation would be a GUI for running the FeaturesReporter scripts.

add_features_script(*rel_script_path*)
Add a features script to run.

add_output_method(*output_method*)
Add an output method

add_sample_source_info(*db_path*, *id*, *ref=False*)

run_json(*backround=False*)

save_json(*out_path='local_json.txt'*)

jade.rosetta_jade.FeaturesJsonCreator.**add_sample_source**(*json_dict*, *sample_source_dict*)

jade.rosetta_jade.FeaturesJsonCreator.**append_scripts_formats_to_json_dict**(*data*, *json_dict*)

jade.rosetta_jade.FeaturesJsonCreator.**initialize_json_dict**(*out_dir*)

jade.rosetta_jade.FeaturesJsonCreator.**run_features_json**(*json_path*, *background=False*, *outpath=''*)
Convenience function Outputs an R script for running a JSON file, and runs it. Works with the new Library structure of the Features Reporter Framework.

jade.rosetta_jade.FeaturesJsonCreator.**run_features_json_old**(*json_path*, *background=False*, *outpath=''*)
Convenience function Run compare_sample_sources with json path.

jade.rosetta_jade.FeaturesJsonCreator.**setup_baseline_scripts_and_formats**(*json_dict*, *type*)

jade.rosetta_jade.FeaturesJsonCreator.**write_json_for_single_recovery_experiment**(*db_path_exp*, *db_path_native*, *exp_id*, *out_path*)
Create a JSON file for recovery of a single experiment.

jade.rosetta_jade.Region module

jade.rosetta_jade.RunRosetta module

class jade.rosetta_jade.RunRosetta.**RunRosetta**(*program=None*, *parser=None*, *db_mode=False*, *json_run=None*)
Bases: `object`

local_run(**args*, ***kwargs*)
Get if we are running locally :rtype: bool

run(**args*, ***kwargs*)

jade.rosetta_jade.RunRosetta.**get_option_strings**(*cmd*)
Get the options as a string to be printed or saved to a file. :param cmd: :rtype: str

jade.rosetta_jade.RunRosetta.**print_full_cmd**(*cmd*, *script_path=None*)

jade.rosetta_jade.RunRosetta.**run_on_qsub**(*cmd*, *queue_dir*, *name*, *print_only=False*, *extra_opts=''*)

jade.rosetta_jade.RunRosetta.**run_on_slurm**(*cmd*, *queue_dir*, *name*, *nodes=False*, *ntasks=None*, *print_only=False*, *extra_opts=''*)

jade.rosetta_jade.RunRosetta.**write_queue_file**(*cmd*, *queue_dir*, *name*)

jade.rosetta_jade.RunRosettaBenchmarks module

class jade.rosetta_jade.RunRosettaBenchmarks.**RunRosettaBenchmarks**(*program=None*, *parser=None*)

Bases: jade.rosetta_jade.RunRosetta.RunRosetta

run()

Run All Benchmarks. This code callse the following:

run -> _get_list_of_benchmarks -> run_benchmark

Returns

run_benchmark(*benchmark_names*, *benchmark_options*)

Run a single benchmark with options.

Parameters

- **benchmark_names** – List of benchmark names
- **benchmark_options** – List of benchmark options

Returns

jade.rosetta_jade.ScoreFiles module

class jade.rosetta_jade.ScoreFiles.**ScoreFile**(*filename*)

get_Dataframe(*scoreterms=None*, *order_by='total_score'*, *top_n=-1*, *reverse=True*)

Get data as a pandas dataframe. Definitely preferred now. :param scoreterms: list :param order_by: str :param top_n: int :param reverse: bool :rtype: pandas.DataFrame

get_decoy_count()

get_decoy_names()

get_ordered_decoy_list(*scoreterm*, *decoy_names=None*, *top_n=-1*, *reverse=False*)

Get an ordered tuple of [[score, decoy_name], ...] Will automatically order some known scoreterms (hbonds_int, dSASA_int)

Return type list[list]

get_score(*decoy*, *scoreterm*)

Get Score of a particular decoy and scoreterm :param decoy: str :param scoreterm: str :rtype: float

get_scores(*scoreterm*, *decoy_names=None*, *top_n=-1*, *reverse=False*)

get_scoreterm(*scoreterm*)

get_scoreterm_names()

get_scoreterms(*scoreterms=''*)

get_stats(*scoreterms=''*, *decoy_names=None*)

jade.rosetta_jade.ScoreFiles.**get_scorefiles**(*indir='/home/docs/checkouts/readthedocs.org/user_builds/bio-jade/checkouts/latest/docs'*)

Get Score files from a directory. Walk through all directories in directory. :param indir: str :rtype: list

jade.rosetta_jade.ScoreFiles.**plot_score_vs_rmsd**(*df*, *title*, *outpath*, *score='total_score'*, *rmsd='looprms'*, *top_p=0.95*, *reverse=True*)

Plot a typical Score VS RMSD using matplotlib, save it somewhere. Return the axes. By default, plot the top 95% :param df: pandas.DataFrame :param outpath: str :param score: str :param rmsd: str :rtype: matplotlib.Axes

jade.rosetta_jade.ScoreFiles.**pymol_session_on_top_df**(*df*, *outdir*, *decoy_dir=None*, *scoreterm='total_score'*, *top_n=10*, *decoy_column='decoy'*, *native_path=None*, *out_prefix_override=None*, *ab_structure=False*, *superimpose=False*, *run_pymol=True*)

Make a PyMol session (or setup a scripter) on top X using a dataframe. Return the scripter for extra control.

df should have an attribute of 'name' or out_prefix_override should be set.

Parameters

- **df** – pandas.DataFrame
- **outdir** – str
- **decoy_dir** – str
- **scoreterm** – str
- **top_n** – int
- **decoy_column** – str
- **native_path** – str
- **out_prefix_override** – str
- **ab_structure** – boolean
- **superimpose** – boolean

Return type PyMolScriptWriter

jade.rosetta_jade.SetupRosettaOptionsBenchmark module

class jade.rosetta_jade.SetupRosettaOptionsBenchmark.**SetupRosettaOptionsBenchmark**(*json_file*)

Bases: jade.rosetta_jade.SetupRosettaOptionsGeneral.SetupRosettaOptionsGeneral

Class for setting up Rosetta Benchmarks. See database/rosetta/benchmark_jsons_rabd/nstruct_test.json for an example.

Basically, a set of benchmarks and rosetta options are given in the JSON. Other keys can be specified for specific benchmarks (like the instructions file stuff in the above file.)

This can be used to use a single JSON file and run RosettaMPI on ALL combinations of benchmarks given.

get_benchmark_names(*only_rosetta=False*)
Get the names of all the benchmarks we will run.

Each benchmark must have a dictionary that defines 'benchmarks' as a list. You may optionally give the rosetta_option. Currently, your subclass of RunRosetta will need to code how all this is run. Hopefully, that will change.

If only_rosetta is true, will only give the benchmark names that are based on rosetta options.

For example:

"outer_cycle_rounds":{ "rosetta_option":"-outer_cycle_rounds", "benchmarks":[25, 50, 75, 100]

},

Return type list

get_benchmarks_of_key(*benchmark_name*)
Get the list of benchmarks for a particular benchmark key. :param benchmark_name: str :rtype: list

get_exp()
Get the benchmark name or fail. :rtype: str

get_non_rosetta_option_benchmark_names()
Similar to get_benchmark_names, but only for options which do not have the tag rosetta_option

Return type list

get_rosetta_option_of_key(*benchmark_name*)
Get the Rosetta option :param benchmark_name: :rtype: str

use_benchmark_for_outdir(*benchmark*)
Should we use the benchmark name for output?

Specified by the 'use_for_outdir' in JSON. If not specified, or benchmark not in list, we assume True!

Parameters **benchmark** – str

Return type bool

use_benchmark_for_prefix(*benchmark*)
Should we use the benchmark name for prefix?

Specified by the 'use_for_prefix' in JSON. If not specified, or benchmark not in list, we assume True!

Parameters **benchmark** – str

Return type bool

jade.rosetta_jade.SetupRosettaOptionsGeneral module

class jade.rosetta_jade.SetupRosettaOptionsGeneral.**SetupRosettaOptionsGeneral**(*cluster_json_file*)
Bases: object

Class for setting up more general Rosetta options for benchmarking and repeatable runs on different clusters. Useful for benchmarking. Subclass for adding more benchmarking settings for specific benchmarks.

get_base_rosetta_flag_string(*indir_root=None*)

Get the full flag string for output. Optionally give indir_root for subclasses that require setting of different directories, but with same root as given in the cluster file. Used primarily for complicated benchmarks.

get_db_mode()

get_indirs()

get_machine_file()

get_nstruct()

get_program()

get_root()

get_xml_script()

get_xml_var_string()

jade.rosetta_jade.alignment module

jade.rosetta_jade.alignment.**align_to_second_pose_save_pdb**(*pose_name*, *pose*, *second_pose*, *outdir*, *overhang=0*, *stem_align=False*)

jade.rosetta_jade.alignment.**get_map_for_rmsd**(*pose*, *second_pose*, *overhang=3*)

jade.rosetta_jade.alignment.**get_mask_for_alignment**(*pose*, *second_pose*, *overhang=0*)
Get mask assuming they are both the same length!

jade.rosetta_jade.alignment.**get_mask_for_stem_alignment**(*pose*, *second_pose*, *stem_size*)

jade.rosetta_jade.alignment.**get_rmsd**(*pose*, *second_pose*, *overhang=0*)
Get RMSD assuming they are both the same length!

jade.rosetta_jade.features module

jade.rosetta_jade.features.**create_features_db**(*pdb_list*, *xml_name*, *compiler*, *score_weights*, *out_db_name*, *out_db_batch*, *outdir*, *use_present_dbs*, *indir=''*, *mpi=True*, *np=5*)
old_db_name = outdir+'/'+out_db_name+'.'+score_weights+".db3" new_db_name = outdir+'/'+out_db_name+'.'+xml_name+'.'+score_weights+".db3" if os.path.exists(old_db_name):

os.system('mv '+old_db_name+' '+new_db_name) print "Old db name already exists. Moving." return

jade.rosetta_jade.features.**rm_features_dbs**(*outdir*, *out_names*)

jade.rosetta_jade.flag_util module

jade.rosetta_jade.flag_util.**get_common_flags_string_for_init**(*flags_name='common_flags.flags'*)
Get a string of common flags as specified in the database. :return: str

jade.rosetta_jade.score_util module

jade.rosetta_jade.score_util.**parse_decoy_scores**(*decoy_path*)
Parse a score from a decoy and return a dictionary. :param decoy_path: :return: defaultdict

jade.utility package

class jade.utility.**vector1**(*seq=()*)
Bases: list

A list indexed at 1!

jade.utility.string_util module

jade.utility.string_util.**deduce_str_type**(*s*)
Deduce the type of a string. Either return the string as the literal, or as the string if not possible. http://stackoverflow.com/questions/13582142/deduce-the-type-of-data-in-a-string

Parameters **s** – str

Returns

CHAPTER 4

Public Apps

4.1 public.antibody_benchmark_utils

4.1.1 RunRosettaBenchmarksMPI.py

This program runs Rosetta MPI locally or on a cluster using slurm or qsub. Relative paths are accepted.

```
usage: RunRosettaBenchmarksMPI.py [-h]
```

4.1.2 bm-RAbD_Jade.py

This program is a GUI used for benchmarking Rosetta Antibody Design.Before running this application, you will probably want to run 'run_rabd_features_for_benchmarks.py to create the databases required.

```
usage: bm-RAbD_Jade.py [-h] [--main_dir MAIN_DIR] [--out_dir OUT_DIR] --jsons
                       [JSONS [JSONS ...]]
```

Named Arguments

--main_dir Main working directory. Not Required. Default = PWD

Default: "/home/docs/checkouts/readthedocs.org/user_builds/bio-jade/checkouts/latest/docs"

--out_dir Output data directory. Not Required. Default = pooled_data

Default: "pooled_data"

--jsons, -j Analysis JSONs to use. See RAbD_MB.AnalysisInfo for more on what is in the JSON.The JSON allows us to specify the final name, decoy directory, and features db associated with the benchmark as well as all options that went into it.

4.1.3 bm-calculate_graft_closure_rabd.py

Calculate the frequence of graft closures.

```
usage: bm-calculate_graft_closure_rabd.py [-h] [--dir DIR] [--outfile OUTFILE]
                                          [--use_ensemble]
                                          [--match_name MATCH_NAME]
```

Named Arguments

--dir, -i	Input directory
--outfile, -o	Path to outfile
--use_ensemble	Use ensembles in calculation Default: False
--match_name	Match a subexperiment in the file name such as relax

4.1.4 bm-calculate_recoveries_and_risk_ratios.py

Calculates and plots monte carlo acceptance values for antibody design benchmarking.

```
usage: bm-calculate_recoveries_and_risk_ratios.py [-h] --jsons
                                                  [JSONS [JSONS ...]]
                                                  [--data_outdir DATA_OUTDIR]
```

Named Arguments

--jsons, -j	Analysis JSONs to use. See RAbD_MB.AnalysisInfo for more on what is in the JSON.The JSON allows us to specify the final name, decoy directory, and features db associated with the benchmark as well as all options that went into it.
--data_outdir, -o	Path to outfile. DEFAULT = data Default: "data"

4.1.5 bm-output_all_clusters.py

Calculates and plots monte carlo acceptance values for antibody design benchmarking.

```
usage: bm-output_all_clusters.py [-h] --jsons [JSONS [JSONS ...]]
                                 [--data_outdir DATA_OUTDIR]
```

Named Arguments

--jsons, -j	Analysis JSONs to use. See RAbD_MB.AnalysisInfo for more on what is in the JSON.The JSON allows us to specify the final name, decoy directory, and features db associated with the benchmark as well as all options that went into it.
--data_outdir, -o	Path to outfile. DEFAULT = data Default: "data"

4.1.6 bm-plot_features.py

Calculates and plots monte carlo acceptance values for antibody design benchmarking.

```
usage: bm-plot_features.py [-h] --jsons [JSONS [JSONS ...]]
                           [--plot_outdir PLOT_OUTDIR]
```

Named Arguments

--jsons, -j Analysis JSONs to use. See RAbD_MB.AnalysisInfo for more on what is in the JSON.The JSON allows us to specify the final name, decoy directory, and features db associated with the benchmark as well as all options that went into it.

--plot_outdir, -p DIR for plots. DEFAULT = plots

Default: "plots"

4.1.7 bm-run_rabd_benchmarks.py

This program runs Rosetta MPI locally or on a cluster using slurm or qsub. Relative paths are accepted.

```
usage: bm-run_rabd_benchmarks.py [-h]
```

4.2 public.antibody_utils

4.2.1 RAbD_Jade.py

GUI application to analyze designs output by RosettaAntibodyDesign. Designs should first be analyzed by both the AntibodyFeatures and CDRClusterFeatures reporters into sqlite3 databases.

```
usage: RAbD_Jade.py [-h] [--db_dir DB_DIR] [--analysis_name ANALYSIS_NAME]
                    [--native NATIVE] [--root_dir ROOT_DIR]
                    [--cdrs [{L1,H1,L1,H2,L3,H3} [{L1,H1,L1,H2,L3,H3} ...]]]
                    [--pyigclassify_dir PYIGCLASSIFY_DIR]
                    [--jsons [JSONS [JSONS ...]]]
```

Named Arguments

--db_dir Directory with databases to compare. DEFAULT = databases

Default: "databases"

--analysis_name Main directory to complete analysis. DEFAULT = prelim_analysis

Default: "prelim_analysis"

--native Any native structure to compare to

--root_dir Root directory to run analysis from

Default: "/home/docs/checkouts/readthedocs.org/user_builds/bio-jade/checkouts/latest/docs"

--cdrs Possible choices: L1, H1, L1, H2, L3, H3

A list of CDRs for the analysis (Not used for Features Reporters)

Default: ['L1', 'L2', 'L3', 'H1', 'H2', 'H3']

--pyigclassify_dir Optional PyIgClassify Root Directory with DBOUT. Used for debugging.

Default: ""

--jsons, -j Analysis JSONs to use. See RAbD_MB.AnalysisInfo for more on what is in the JSON.The JSON allows us to specify the final name, decoy directory, and features db associated with the benchmark as well as all options that went into it.

4.2.2 convert_IMGT_to_fasta.py

This script converts an IMGT output file (5_AA-seqs.csv) to a FASTA. All Framework and CDRs are concatonated. * is skipped. The FASTA file can then be used by PyIgClassify.

```
usage: convert_IMGT_to_fasta.py [-h] --inpath INPATH --outpath OUTPATH
```

Named Arguments

--inpath, -i Input IMGT file path

--outpath, -o Output Fasta outfile path.

4.2.3 create_features_json.py

This script will create either cluster features or antibody features json for use in Features R script. Example Cmd-line: python create_features_json.py –database databases/baseline_comparison.txt –scripts cluster

```
usage: create_features_json.py [-h] [--databases [DATABASES [DATABASES ...]]]
                               [--script {cluster,antibody,interface,antibody_minimal}
↪]
                               [--db_path DB_PATH] [--outdir OUTDIR]
                               [--outname OUTNAME]
                               [--add_comparison_to_this_json ADD_COMPARISON_TO_THIS_
↪JSON]
                               [--run]
```

Named Arguments

--databases, -l List of dbs: db_name,short_name,ref keyword if the reference databaseSeparated by white space.

Default: []

--script, -s Possible choices: cluster, antibody, interface, antibody_minimal

Script type. Will setup the appropriate output formats and R scripts

Default: "antibody_minimal"

--db_path, -p Path to databases. Default is pwd/databases

Default: "/home/docs/checkouts/readthedocs.org/user_builds/bio-jade/checkouts/latest/docs/databases"

--outdir, -o Where to put the result of the analysis scripts. Currently unsupported by the features framework.

Default: "/home/docs/checkouts/readthedocs.org/user_builds/bio-jade/checkouts/latest/docs/plots"

--outname, -n Output file name of json file

Default: "local_json_compare_ss.json"

--add_comparison_to_this_json, -a Add all this data to this json as more sample sources.

--run, -r Go ahead and run compare_sample_sources.R. Must be in path!!

Default: False

4.2.4 generate_rabd_features_dbs.py

Generates RAbD Features DBs using RunRosettaMPI in db mode.

```
usage: generate_rabd_features_dbs.py [-h]
```

4.2.5 match_antibody_structures.py

This App aims to make pymol alignments using the PyIgClassify database and structures, matching specific criterion.

```
usage: match_antibody_structures.py [-h] --db DB --ab_dir AB_DIR --where WHERE
                                    [--outdir OUTDIR] [--prefix PREFIX]
                                    [--cdr CDR] [--native NATIVE]
```

Required Arguments

--db, -d Database to use from PyIgClassify.

--ab_dir, -b Directory with renumbered antibody PDBs (Full or CDRs-only)

--where, -w Your where clause for the db in quotes. Not including WHERE. Use ' ' for string matches

Other Arguments

--outdir, -o Output directory.

Default: "/home/docs/checkouts/readthedocs.org/user_builds/bio-jade/checkouts/latest/docs"

--prefix, -p Output prefix

--cdr, -c Optionally load the CDR PDBs of the given type in the ab_dir. If this option is set, the ab_dir should be of CDRs only from PyIgClassify.

--native, -n Align everything to this PDB, the native or something you are interested in.

4.2.6 order_ab_chains.py

Reorders PDBFiles in a dirctory according to A_LH in order for Rosetta Antibody Design benchmarking. Removes HetAtm

```
usage: order_ab_chains.py [-h] [--in_dir IN_DIR] [--in_pdblist IN_PDBLIST]
                          [--in_single IN_SINGLE] [--out_dir OUT_DIR]
                          [--reverse]
```

Named Arguments

--in_dir, -i Input Directory of PDB files listed in any passed PDBLIST. Default=PWD

Default: "/home/docs/checkouts/readthedocs.org/user_builds/bio-jade/checkouts/latest/docs"

--in_pdblist, -l Input PDBList file. Assumes PDBList has no paths and requires an input directory as if we run Rosetta.

Default: ""

--in_single, -s Path to Input PDB File, instead of list.

Default: ""

--out_dir, -d Output Directory. Resultant PDB files will go here.

Default: "reordered"

--reverse, -r Reverse order (LH_A instead of A_LH). Used for snugdock

Default: False

4.2.7 split_antibody_components.py

Script for splitting AHO renumbered antibodies into Fv, Fc, and linker regions

```
usage: split_antibody_components.py [-h] [--any_structure] --ab_dir AB_DIR
                                    --output_dir OUTPUT_DIR
```

Named Arguments

--any_structure Be default, we only output structures with both L/H. Pass this option to split structures that are L or H only.

Default: False

--ab_dir, -a Antibody Directory with AHO-renumbered structures to split. Can be .pdb, or .pdb.gz

--output_dir, -o Output Directory for antibody structures.

4.3 public.general

4.3.1 canceljobs.py

Call scancel to cancel a consecutive set of cluster job numbers

```
usage: canceljobs.py [-h]
```

4.3.2 convert_fig.py

Converts images to TIFF figures at 300 DPI for publication using sips. Arguments: INFILE OUTFILE

```
usage: convert_fig.py [-h]
```

4.3.3 genscript_to_fasta.py

This script outputs fasta files from a genscript format. Pass the –format option to control which genscript format as input ~~~ Ex: python genscript_mut_to_fasta.py –format mutagenesis MutagenesisFormatU68 ~~~

```
usage: genscript_to_fasta.py [-h] --format {mutagenesis,GeneSynth} infile
```

Positional Arguments

infile The mutagenesis format file.

Named Arguments

--format Possible choices: mutagenesis, GeneSynth

The genscript file format

4.3.4 get_seq.py

Uses Biopython to print sequence information. Example: get_seq.py –pdb 2j88_A.pdb –format fasta –outpath test.txt

```
usage: get_seq.py [-h] [--pdb PDB] [--pdblist PDBLIST]
                  [--pdblist_input_dir PDBLIST_INPUT_DIR] [--chain CHAIN]
                  [--cdr CDR]
                  [--format {basic,fasta,general_order,IgG_order,IgG_order_lambda,IgG_
↪order_kappa,IgG_order_heavy}]
                  [--outpath OUTPATH] [--prefix PREFIX] [--region REGION]
                  [--strip_c_term STRIP_C_TERM] [--pad_c_term PAD_C_TERM]
                  [--output_original_seq]
```

Named Arguments

--pdb, -s Input PDB path

--pdblist, -l Input PDB List

--pdblist_input_dir, -i Input directory if needed for PDB list

--chain, -c A specific chain to output

Default: “”

--cdr Pass a specific CDR to output alignments of.

Default: “”

--format Possible choices: basic, fasta, general_order, IgG_order, IgG_order_lambda, IgG_order_kappa, IgG_order_heavy

The output format requried.

Default: “fasta”

--outpath, -o Output path. If none is specified it will write to screen.

--prefix, -t Tag to add before chain

Default: “”

--region specify a particular region, start:end:chain

--strip_c_term Strip this sequence off the C-term of resulting sequences. (Useful for antibodies

--pad_c_term Pad this sequence with some C-term (Useful for antibodies

--output_original_seq Output the original sequence and the striped seqeunce if stripped. Default FALSE.

Default: False

4.3.5 rename_designs.py

Renames original files to new names for design ordering. Copy all models going to be ordered into a single directory first. Run from directory with pdb files already copied in!

```
usage: rename_designs.py [-h] -i NEW_NAMES
```

Named Arguments

-i, --new_names File with new to old names. Example line: new_name * filename. Can have lines that don’t have all three. Will only rename if it has a star in the second column.

4.4 public.pdb_utils

4.4.1 place_TERs.py

This script places ters between ATOM/HETATM columns. This is currently needed to reload symmetrized glycan posescreated by the god aweful make_symm_file.pl Rosetta script. USE: place_TERs.py my_pdb - Does it in place.

```
usage: place_TERs.py [-h] [pdb_files [pdb_files ...]]
```

Positional Arguments

pdb_files Path to PDB files we will be stripping.

4.4.2 strip_ANISOU.py

Strips ANISOU lines out of PDBs.

```
usage: strip_ANISOU.py [-h] [pdb_files [pdb_files ...]]
```

Positional Arguments

pdb_files Path to PDB file we will be stripping.

4.4.3 strip_ter.py

This simple script strips ters out of a PDB file and overwrites the input. PyMol places ters when th numbering is not 1-1. And then Rosetta will F your Shit up.

```
usage: bstrip_ter.py [-h] [pdb_files [pdb_files ...]]
```

Positional Arguments

pdb_files Path to PDB file we will be stripping.

4.5 public.pyrosetta

4.5.1 build_loop_pyrosetta.py

This script builds a loop between two places in a structure with the given sequence, and closes the loop.It is not meant to be the last modeling step, just to create missing density or to prepare for loop modeling.

```
usage: build_loop_pyrosetta.py [-h] --start START --stop STOP --sequence
                               SEQUENCE [--out_prefix OUT_PREFIX]
                               [--retain_aligned_roots] --pdb PDB [--kic]
                               [--dump_midpoints]
```

Named Arguments

--start Starting resnum. Ex: 24L

--stop Ending resnum. Ex. 42L.

--sequence Sequence of the loop

--out_prefix Any prefix to give results.

Default: "**loop_built_**"

--retain_aligned_roots Attempt to keep any aligned root residues during the build

Default: False

--pdb, -s Input model

--kic Run KIC peruturber after closing the loop?

Default: False

--dump_midpoints Dump midpoint PDBs?

Default: False

4.5.2 find_my_glycans.py

This app is the PyRosetta equivalent of GlycanInfo. Print carbohydrate info about the pose. Pass the pose in as an argument

```
usage: find_my_glycans.py [-h]
```

4.5.3 find_my_residues.py

Simple app to scan a PDB file and print PDB info and Rosetta understood chains and resnums.

```
usage: find_my_residues.py [-h] [--chain CHAIN] [--echo_input] pdb_file
```

Positional Arguments

pdb_file The PDB file to scan.

Named Arguments

--chain, -c Specify only a single chain to scan.

--echo_input, -e Echo the input structure as output. This is to check how Rosettta worked reading it.

Default: False

4.5.4 get_mutation_energy.py

Basic app to get mutation energy of each residue in a particular region using PyRosetta

```
usage: get_mutation_energy.py [-h] [--pdb PDB] [--outpath OUTPATH]
                              [--filename FILENAME] [--region REGION]
                              [--relax_whole_structure] [--alanine_scan]
```

Named Arguments

--pdb, -s Path to PDB file. Required.

--outpath, -o Full output directory path. Default is pwd/RESULTS

Default: "/RESULTS"

--filename, -n The filename of the results file

Default: "mutation_energies.txt"

--region, -r (region designated as start:end:chain) If none is given, will use whole PDB

--relax_whole_structure, -m Relax the whole structure? Default is to only relax chain under question. If no region is set, will default to true

Default: False

--alanine_scan, -a Trigger the script to do an alanine scan of the mutations instead of a full mutational scan.

Default: False

4.6 public.rosetta

4.6.1 score_analysis.py

This utility parses and extracts data from score files in JSON format

```
usage: score_analysis.py [-h] [-s [SCORETYPES [SCORETYPES ...]]] [-n TOP_N]
                         [--top_n_by_10 TOP_N_BY_10]
                         [--top_n_by_10_scoretype TOP_N_BY_10_SCORETYPE]
                         [--decoy_names [DECOY_NAMES [DECOY_NAMES ...]]]
                         [--list_scoretypes] [--pdb_dir PDB_DIR] [--summary]
                         [--csv] [--make_pdblist] [--pymol_session]
                         [--plot [PLOT [PLOT ...]]] [--copy_top_models]
                         [--prefix PREFIX] [--outdir OUTDIR]
                         [--plot_type {line,scatter,bar,hist,box,kde,area,pie,hexbin}]
                         [--plot_filter PLOT_FILTER] [--native NATIVE]
                         [--ab_structure] [--super SUPER]
                         [scorefiles [scorefiles ...]]
```

Positional Arguments

scorefiles A list of scorefiles

Named Arguments

-s, --scoretypes List of score terms to extract

Default: ['dSASA_int', 'delta_unsatHbonds', 'hbonds_int', 'total_score', 'dG_separated', 'top_n_by_10']

-n, --top_n Only list Top N when doing top scoring decoys or making pymol sessionsDefault is to print all of them.

Default: -1

--top_n_by_10 Top N by 10 percent total score to print out.

Default: 10

--top_n_by_10_scoretype Scoretype to use for any top N by 10 printing. If scoretype not present, won't do anything.

Default: "dG_separated"

--decoy_names Decoy names to use

Default: []

--list_scoretypes List score term names

Default: False

--pdb_dir, -d Directory for PDBs if different than the directory of the scorefile

OUTPUT

General output options.

--summary, -S Compute stats summarizing data

Default: False

--csv, -c Output selected columns, top, and decoys as CSV.

Default: False

--make_pdblist Output PDBlist file(s)

Default: False

--pymol_session Make pymol session(s) of the scoretypes specified

Default: False

--plot Plot one score type vs another. Save the plot. 2 or 3 Arguments. [X, Y, 'Title'] OR [X, 'Title']. If title has spaces, use quotes. Nothing special, just used for quick info.

Default: []

--copy_top_models Copy the top -n to the output directory for each scorefile passed.

Default: False

--prefix, -p Prefix to use for any file output. Do not include any _

Default: ""

--outdir, -o Output dir. Default is current directory.

Default: "/home/docs/checkouts/readthedocs.org/user_builds/bio-jade/checkouts/latest/docs"

PLOTTING

Options for plot output

--plot_type Possible choices: line, scatter, bar, hist, box, kde, area, pie, hexbin

The type of plot we are outputting.

Default: "scatter"

--plot_filter Filter X to top Percent of this - useful to remove outliers.

Default: 1.0

PYMOL

Options for pymol session output

--native Native structure to use for pymol sessions.

--ab_structure Specify if the module is a renumbered antibody structure. Will run pymol script for ab-specific selection

Default: False

--super Super this selection instead of align all to.

4.6.2 RunRosettaMPI.py

This program runs Rosetta MPI locally or on a cluster using slurm or qsub. Relative paths are accepted.

```
usage: RunRosettaMPI.py [-h]
```

4.6.3 RunRosettaDBMode.py

This program runs Rosetta MPI locally or on a cluster using slurm or qsub. Relative paths are accepted.

```
usage: RunRosettaDBMode.py [-h]
```

4.6.4 util

4.6.5 check_missing_rosetta_nstruct.py

This extremely simple script checks nstruct of the input files and outputs which nstruct number is missing.

```
usage: check_missing_rosetta_nstruct.py [-h] [-n NSTRUCT]
                                        [--pdb_files [PDB_FILES [PDB_FILES ...]]]
                                        [--pdblist PDBLIST] [--dir DIR]
```

Named Arguments

-n, --nstruct Default: 1000

--pdb_files Path to PDB files we will be checking.

--pdblist, -l Optional INPUT PDBLIST (without 00s, etc. for which to check

--dir The Directory to check. As opposed to a list of pdb files.

4.6.6 create_score_json_from_scored_decoys.py

This script creates a Rosetta score file from a set of structures - by parsing the score from them. Pass a directory, a PDBLIST, and/or a list of filenames

```
usage: create_score_json_from_scored_decoys.py [-h] [--prefix PREFIX]
                                               [decoys [decoys ...]]
```

Positional Arguments

decoys A directory, a PDBLIST, and/or a list of filenames

Default: []

Named Arguments

--prefix Any prefix to use.

Default: “”

4.6.7 insert_natives_table_into_features_db.py

This script takes a PDBLIST of natives and then adds a new table to the database with struct_id as proper foreign primary key and the native structure based solely on a search of the name tag.

```
usage: insert_native_table_into_features_db.py [-h] [--pdblist PDBLIST]
                                               [--db DB]
```

Named Arguments

--pdblist PDBLIST of native structures used.

--db The database we are working on.

CHAPTER 5

Pilot Apps

5.1 apps.pilot.jadolfbr

5.1.1 copy_top_each_strategy.py

```
usage: copy_top_each_strategy.py [-h] [-n N] -i INDIR -o OUTDIR
                                 [-s [STRATEGIES [STRATEGIES ...]]]
```

Named Arguments

-n	Number of models to copy. DEFAULT = 2
	Default: 2
-i, --indir	Input directory
-o, --outdir	Output directory
-s, --strategies	The type of strategies we are interested in
	Default: ['delta_unsats_per_1000_dSASA', 'dG_top_Ptotal']

5.1.2 glycan_basic_LCM_protocol.py

```
usage: glycan_basic_LCM_protocol.py [-h] --infile INFILE
                                    --glycosylation_position
                                    GLYCOSYLATION_POSITION
                                    [--glycosylation_name GLYCOSYLATION_NAME]
                                    [--nstruct NSTRUCT] [--cycles CYCLES]
```

Named Arguments

--infile, -s Input PDB.

--glycosylation_position, -g Glycosylation site. Rosetta resnum or resnumChain, ex: 463G

--glycosylation_name, -n Glycosylation name

Default: “man5”

--nstruct Number of output structures

Default: 1

--cycles, -c Total number of cycles to attempt using the LCM

Default: 75

CHAPTER 6

Indices and tables

- genindex
- modindex
- search

Python Module Index

a

j

Index

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W