AESOP Documentation

Release v1.0.1

Reed Harrison, Rohith Mohan, Dimitrios Morikis

Apr 25, 2018

Contents

1 A python library for studying electrostatics in proteins 1

2 Project summary 3

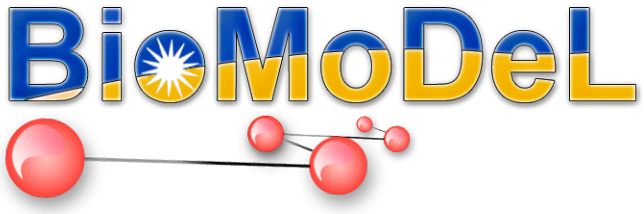
Bibliography 45

Python Module Index 47

CHAPTER 1

A python library for studying electrostatics in proteins

AESOP is developed and maintained by members of the Biomolecular Modeling and Design Lab at the University of California, Riverside, including: Reed Harrison, Rohith Mohan, and Dimitrios Morikis.



Indices and tables

- genindex
- modindex
- search

CHAPTER 2

Project summary

For many proteins complexes, long range electrostatic interactions play a pivotal role in the formation of the encounter complex. Charge-charge interactions can also serve to thermodynamically stabilize any resulting complex. In many cases, however, optimization of a protein associating with its binding partner is sufficient to increase protein activity as many protein systems are diffusion limited. The AESOP framework provides a tool to investigate the electrostatic nature of protein interactions both across protein families as well as within indivdiual proteins in terms of individual amino acid contributions.

AESOP is implemented in Python 2.7 and depends on a number of computational tools including: APBS, PDB2PQR, ProDy, and Modeller. Documentation can be found at aesop.readthedocs.io. We welcome questions and collaboration on the GitHub page for AESOP at https://github.com/rohithmohan/aesop.

2.1 Methods

AESOP currently supports three computational methods:

- **Alascan**
  - Perform a computational alanine scan on a provided protein structure using a side-chain truncation scheme
  - Association free energies for mutatants (relative to the parent) may be predicted if 2 or more selection strings are provided
  - Users may restrict mutations to some region of the protein structure
- **DirectedMutagenesis**
  - Perform a directed mutagenesis scan on a provided protein structure using Modeller to swap amino acids
  - Association free energies for mutatants (relative to the parent) may be predicted if 2 or more selection strings are provided
  - Mutations must be specified

- **ElecSimilarity**
  - Compare electrostatic potentials of multiple protein structures, or compare electrostatic potentials of alanine scan mutants
  - If structures are very dissimilar, the user should superpose coordinates for each protein structure according to their desired method

2.2 General Utilities

In addition to the three main computational methods, AESOP provides a total of three functions capable of displaying results as figures during interactive Python sessions or saving results as figures during any Python script:

- **aesop.plotScan()**
  - Show bargraph summary of results from computational mutagenesis methods (Alascan, DirectedMutagenesis)
- **aesop.plotNetwork()**
  - Show network summary of results from computational mutagenesis methods (Alascan, DirectedMutagenesis)
- **aesop.plotESD()**
  - Show heatmap summary of results from methods exploring electrostatic similarity (ElecSimilarity)
- **aesop.plotDend()**
  - Show dendrogram summary of results from methods exploring electrostatic similarity (ElecSimilarity)
- **aesop.plotScan_interactive()**
  - Show bargraph summary of results from computational mutagenesis methods in a more interactive format (Alascan, DirectedMutagenesis)
- **aesop.plotNetwork_interactive()**
  - Show network summary of results from computational mutagenesis methods in a more interactive format (Alascan, DirectedMutagenesis)
- **aesop.plotESD_interactive()**
  - Show heatmap summary of results from methods exploring electrostatic similarity in a more interactive format (ElecSimilarity)
- **aesop.writePDB()**
  - Save PDB file with changes in free energy of association relative to the parent (by residue) in the beta-factor column

2.3 Notes

- We recommend using Anaconda to aid in installation of Python scientific libraries
- Depending on your platform, ProDy may need to be installed with an executable

2.4 Contents

2.4.1 About

AESOP is a computational framework to investigate electrostatic interactions that promote association of protein complexes and to compare similarity of electrostatic potentials across families of proteins. In the former case, protein engineering (or re-engineering) of enzymes or other industrial proteins may benefit from AESOP by using the software to optimize association of some protein with its binding partner. In such applications, the activity of the protein system is diffusion-limited. By promoting association, the protein may be enhanced for its application.

Acknowledgements

AESOP is developed in the Biomolecular Modeling and design Lab (BioMoDeL) under the supervision of Professor Dimitrios Morikis. The python library of AESOP is developed by Reed Harrison and Rohith Mohan *[Harrison2016]*, and is based on the original AESOP framework, written in R, that was developed and parametrized by Chris Kieslich and Ronald Gorham *[Kieslich2011-1] [Gorham2011-1] [Gorham2011-2] [Kieslich2011-2]*. A preliminary computational protocol for electrostatic similarities that preceded AESOP was developed by Jianfeng Yang.

The AESOP acronym stands for Analysis of Electrostatic Structures Of Proteins. The original AESOP acronym stood for Analysis of Electrostatic Similarities Of Proteins.

References

Published applications of AESOP

The following references are examples of analyses that AESOP can perform.

2.4.2 Installation

AESOP is a Python 2.7 library, and is not compatible with Python 3.

Anaconda installation

We recommend installing the Anaconda Python Distribution as it comes with several packages used by AESOP pre-installed. Anaconda can be downloaded from here. If you choose this route, install Anaconda2. Anaconda3 is not compatible with AESOP.

> **Caution:** The 64-bit version is recommended for Linux, Mac OS, and Windows. To install ProDy in 64-bit windows, you can use an installer from PyPI. If you have issues with a 64-bit installation, AESOP should be compatible with a 32-bit Python installation, though it is not as well tested.

Install PDB2PQR, APBS and Coulomb

Please download and install the appropriate version of PDB2PQR and APBS (Coulomb installs alongside APBS) for your operating system. Ensure that the executables are added to your path by running the executable in terminal/command prompt:

```
apbs

pdb2pqr
```

Note: **Linux users:** Add the directories containing the apbs, pdb2pqr, and coulomb executables to path in your `~/.profile`. As an example:

```
export PATH=/opt/pdb2pqr-linux-bin64-2.1.0:$PATH
export PATH=/opt/APBS-1.4.2.1-linux64/bin:$PATH
export PATH=/opt/APBS-1.4.2.1-linux64/share/apbs/tools/bin:$PATH
```

You may encounter issues with PDB2PQR if you are using an older version that uses the `pdb2pqr.py` script. In this case, either update to a newer binary PDB2PQR distribution, or specify the full path to `pdb2pqr.py` in AESOP.

Mac users: If you are using the APBS binary/executable to install (as opposed to compiling from source), you may need to add the following line to your path:

```
export PATH=$PATH:`dirname '/Applications/APBS.app/Contents/MacOS/apbs_term'`
```

Then you may add the pdb2pqr executable to path as in the Linux case

Windows users: Navigate to system settings (Control panel > System and Security > System) and click advanced settings. Select the `environment variables` box and edit the `Path` variable for the system or for the user. Append the paths to the directories containing the apbs, pdb2pqr, and coulomb executables (each as a separate entry in the Path variable).

Install other dependencies

ProDy:

```
pip install prody
```

Note: If pip is unable to install prody (which is likely to happen if you are running a Windows operating system), then you may download installation materials appropriate for your OS directly from the PyPI website. For Windows, it is typically easier to use the appropriate installation executable appropriate for your Python 2.7 installation (choose amd64 for 64-bit Python 2.7 and choose win32 for 32-bit Python 2.7). As a reminder, AESOP is only compatible with Python 2.7.

Modeller:

```
conda config --add channels salilab
conda install modeller
```

Note: Modeller will require users to have a license key. Registration is located at the Sali Lab Website. The Modeller license key will be used in the following manner:

Edit //anaconda/lib/modeller-9.17/modlib/modeller/config.py and replace XXXX with your Modeller license key (or set the KEY_MODELLER environment variable before running 'conda install').

Multiprocessing (optional):

```
pip install multiprocessing
```

Note: Multiprocessing is only needed if you are planning to utilize multiple cores in your analysis.

NetworkX (optional):

```
conda install networkx
```

Note: NetworkX is only required for the `plotNetwork` utility.

Plotly (optional):

```
pip install plotly
```

Note: Plotly is only required for the interactive plotting functions such as `plotScan_interactive`, `plotNetwork_interactive`, and `plotESD_interactive`.

Installing AESOP

You can install AESOP from PyPi using:

```
pip install aesop
```

If you already have it installed, you can upgrade to the latest version using:

```
pip install --upgrade aesop
```

If you are having issues installing through PyPi, you may try to *Install from source*.

Install from source

To install from source, you can use the following commands to clone the GitHub repository and install manually:

```
git clone git@github.com:rohithmohan/aesop.git
cd aesop-python
pip install setup.py
```

Note: This may require administrative privileges.

2.4.3 Preparing PDB Files

AESOP requires protein structures that comply with the PDB format. Given a structure from the Protein Data Bank, the user needs to consider that coordinates for some atoms may not be resolved in the deposited structure. Thus, residues may be missing from a protein even though the sequence may be known. To fix such issues, the user must perform homology modelling to model and refine gaps. A number of computational tools exist to add these missing residues including Modeller, PDBFixer, UCSF Chimera, Pymol, and SWISS-MODEL. If you are willing to install

OpenMM and all other libraries within the OMNIA channel, then PDBFixer may be installed through anaconda quite easily and offers a simple graphical user interface for PDB preparation.

If the PDB contains all residues but is missing a few atoms in one or more residues, AESOP has a function that will call complete_pdb from Modeller to fill in missing atoms. You may use the function as follows:

```
from aesop import complete_structure

pdbfile = 'input.pdb'
outfile = 'output.pdb'

complete_structure(pdb=pdbfile, dest=outfile, disu=False)
```

The above snipped of code reads `input.pdb`, fills in missing atoms (not missing residues), and writes the completed structure to `output.pdb`. If `input.pdb` has disulfide bridges, simply set `disu=True` to predict and patch disulfides.

While AESOP will handle protein structures where residue numbering overlaps between protein chains, we advise users to make sure only a single model is present in the PDB file to prevent unforseen complications. Additionally, each chain should be represented by a unique identifier that complies with the PDB format.

2.4.4 Atomic Selections

In order to allow for advanced atomic selections with protein structures, AESOP utilizes selection macros from ProDy. Using these macros, the end user can easily string together boolean statements based on protein chains, residue numbers, amino acid, atom name, physicochemical properties, or even distance criteria that describe the portion of the protein structure file to select. For more examples and explanations concerning selection strings please see the ProDy webpage for atomic selections.

Basic Examples

Selection string for chain A of a PDB file:

```
'chain A'
```

Selection string for chain A and residue numbers 1 to 100:

```
'chain A and resnum 1 to 100'
```

Selection string for chain A and protein:

```
'chain A and protein'
```

Selection for chain A or chain C:

```
'chain A or chain C'
```

Alanine scan example

For the alanine scan, we suggest each element of the `selstr` list contains a separate chain:

```
selstr = ['chain A', 'chain B', 'chain C']
```

Ideally, all chains used should comprise a single protein complex. In case the protein complex is quite large and only a handful of mutations are of interest to the end-user, then the region argument may be used to restrict `selstr` to some subset of residues. For this reason, `region` should be a list of the same length as `selstr`. In the current example, if we only wish to mutate residues within 10 angstroms of chain C, then you could specify `region` in the following manner:

```
region = ['within 10 of chain C', 'within 10 of chain C', 'within 10 of chain C']
```

Once again, for in depth discussion of more complicated selection strings, please refer to the ProDy website.

DirectedMutagenesis scan example

For the directed mutagenesis scan, the user must specify the subunits of the protein complex (`selstr`), the targeted residue(s) to mutate (`target`), and mutation to perform (`mutation`). As in the alanine scan, we suggest each element of `selstr` to contain a separate chain of the protein complex:

```
selstr = ['chain A', 'chain B']
```

In order to specify targeted residues to mutate, each element of `target` must contain all residues that will be mutated. Since residue numbers may overlap between chains of the protein structre, the user may need to additionally specify a chain. For example:

```
target = ['resnum 50', 'resnum 50 in chain B', '(resnum 50 or resnum 60) and chain B']
```

In the first element ('resnum 50), all residues with residue number 50 will be mutated. In the second element ('resnum 50 in chain B'), only the residue with number 50 in chain B will be mutated. In the third element ('(resnum 50 or resnum 60) and chain B'), only residues numbered 50 or 60 in chain B will be mutated.

Next, the user must specify how to mutate each element of `target` by specifying a 3 letter amino acid code for each element of the target. These codes should be stored in a list (here, we use the variable name `mutation`):

```
mutation = ['ALA', 'ARG', 'ASP']
```

Since each element of `target` corresponds to each element of `mutation`, the mutations specified above will perform several different mutations. Namely, residues selected by the first element of `target` will be mutated to alanine; residues selected by the second element of `target` will be mutated to arginine; and residues selected by the third element of `target` will be mutated to aspartatic acid. Currently AESOP does not support mutation of two amino acids to two different amino acids simultaneously, though this may be added as a feature in the future. In general, we prefer to mutate one amino acid at a time to prevent significantly changing the structure of the native protein throughout the analysis.

2.4.5 Electrostatic Similarity

The electrostatic similarity method generates grid potentials for a list of PDB files and compares all potential files in a pairwise manner. Here we will provide test cases that compares several members of a family of plant proteins. You may download all necessary files for these examples at this link: `download`. These examples are based on a more comprehensive, published study *[Chae2010]*.

ElecSimilarity(pdbfiles[, pdb2pqr_exe, ...])	Summary

Note: The `ElecSimilarity` method should only be used to compare structurally and functionally similar proteins. Additionally, all protein structure file should be superposed in a consistent grid space. While the method implements a

superpositioning algorithm from Modeller, the user should verify that the final structures are suitably superposed. For some applications, users may aquire better results with a different superpositioning scheme. These structure files can be found in the `pdb_files` or `pqr_files` folder within the job directory.

Example case 1: LTP plant proteins

Open a new python session, import the ElecSimilarity class, and import the plotDend and plotESD functions:

```
from aesop import ElecSimilarity, plotDend, plotESD
```

Warning: If you are planning to leverage multiple CPU threads for a faster analysis, please know that extra steps may be required. Specifically, you must protect the entry point of the program according to multiprocessing documentation. You may do this by putting the following code at the beginning of your Python script:

```
if __name__ == '__main__':
    # place remaining code here and maintain level of indentation
```

This precaution becomes unnecessary if you are running the analysis inside an interactive Python session. In the downloadable zip files, we have already placed this protection in the run script, so that you may run the analysis as follows in your platform's terminal:

```
python run_ltp_ex1.py
```

Failure to protect the entry point may result in an infinite loop of process spawning.

Next, you must specify the full paths to your `apbs` and `pdb2pqr` executables, if the paths for the directories containing the executables have not already been added to the environment. Here is an example for a Windows system:

```
path_apbs    = 'C:\\APBS\\apbs.exe'
path_pdb2pqr = 'C:\\PDB2PQR\\pdb2pqr.exe'
```

Now we will specify what PDB files the method should compare. Here we will use only 3 PDB files (`download`). After downloading the PDB files, unzip them and place them in the current working directory:

```
pdbfiles = ['1MZL.pdb', 'SCA1.pdb', 'SCA3.pdb']
```

Warning: If you are using your own PDB, make sure the PDB contains no missing heavy atoms. Consider also removing non-standard amino acids. PDBFixer is one option for cleaning PDB files in preparation for AESOP.

Note: If you only provide a single PDB file, AESOP will generate a library of mutants by side-chain truncation as in the Alascan class. You can force the ElecSimilarity class to generate mutants for all structures by specifying a list of selection strings that describe all regions of the PDB to mutate.

When the method is run, intermediate files will be generated and stored in a folder of the current working directory. The user has the option of naming this folder by specifying a job name:

```
jobname = 'LTP_test1'
```

Next, the method is initialized by:

```
family = ElecSimilarity(pdbfiles=pdbfiles,
                        pdb2pqr_exe=path_pdb2pqr,
                        apbs_exe=path_apbs,
                        jobname=jobname)
```

Finally, we are ready to run the analysis. To superpose structures before running, set superpose to True (please not that this superpositioning algorithm requires the Modeller library). To center structures before running, set center to True. Ideally, the end user should ensure that all PDB structures have consistent coordinates. This analysis will take several minutes, so please be patient:

```
family.run(superpose=True, center=False)
```

If you are running your analysis on a workstation and want to parallelize the calculation, then you may do so as follows:

```
family.run_parallel(superpose=True, center=False)
```

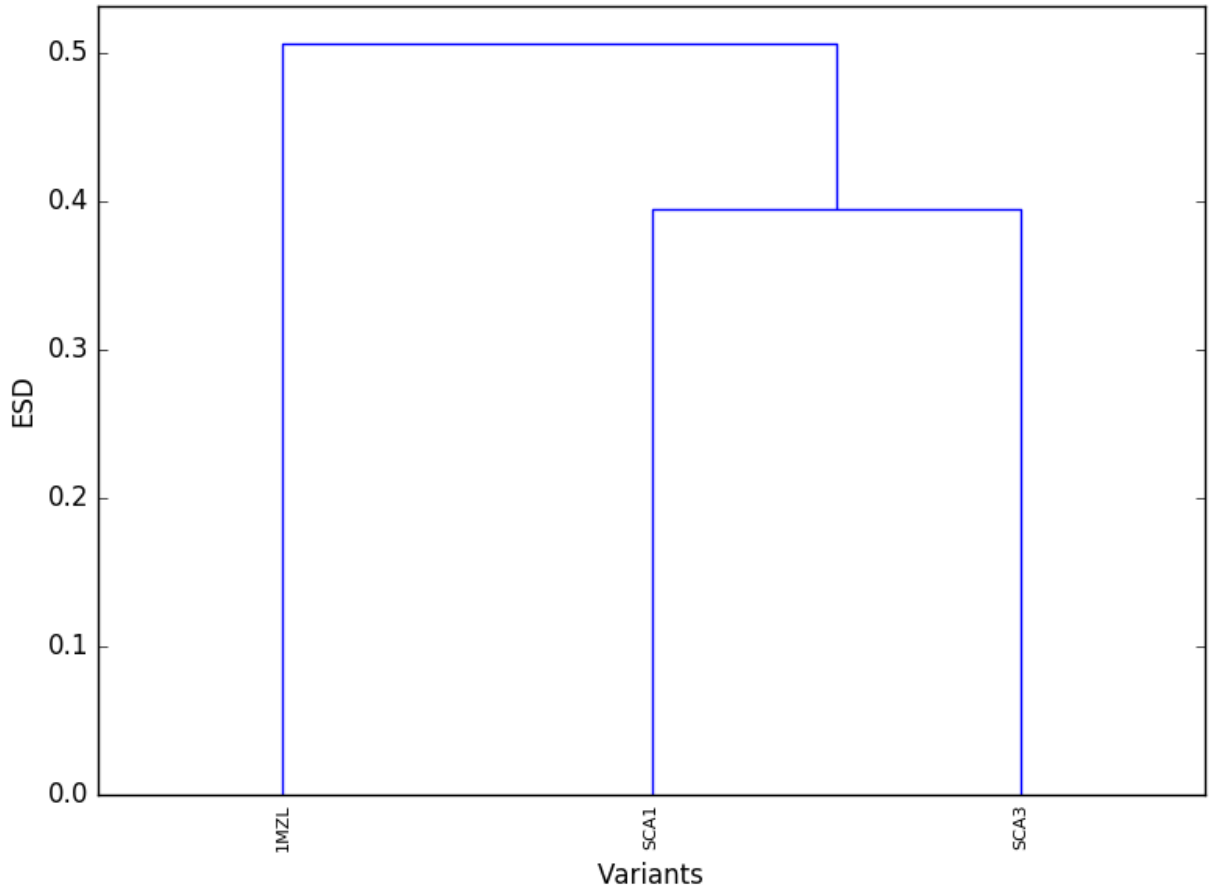
After the run is complete, AESOP will report if any Warnings or Errors were detected in APBS or PDB2PQR. The full logs are stored in the family.logs and can be viewed or written to file in the following manner:

```
family.viewLogs()
family.writeLogs(filename="family_logs.txt")
```

You can view results using built-in functions:

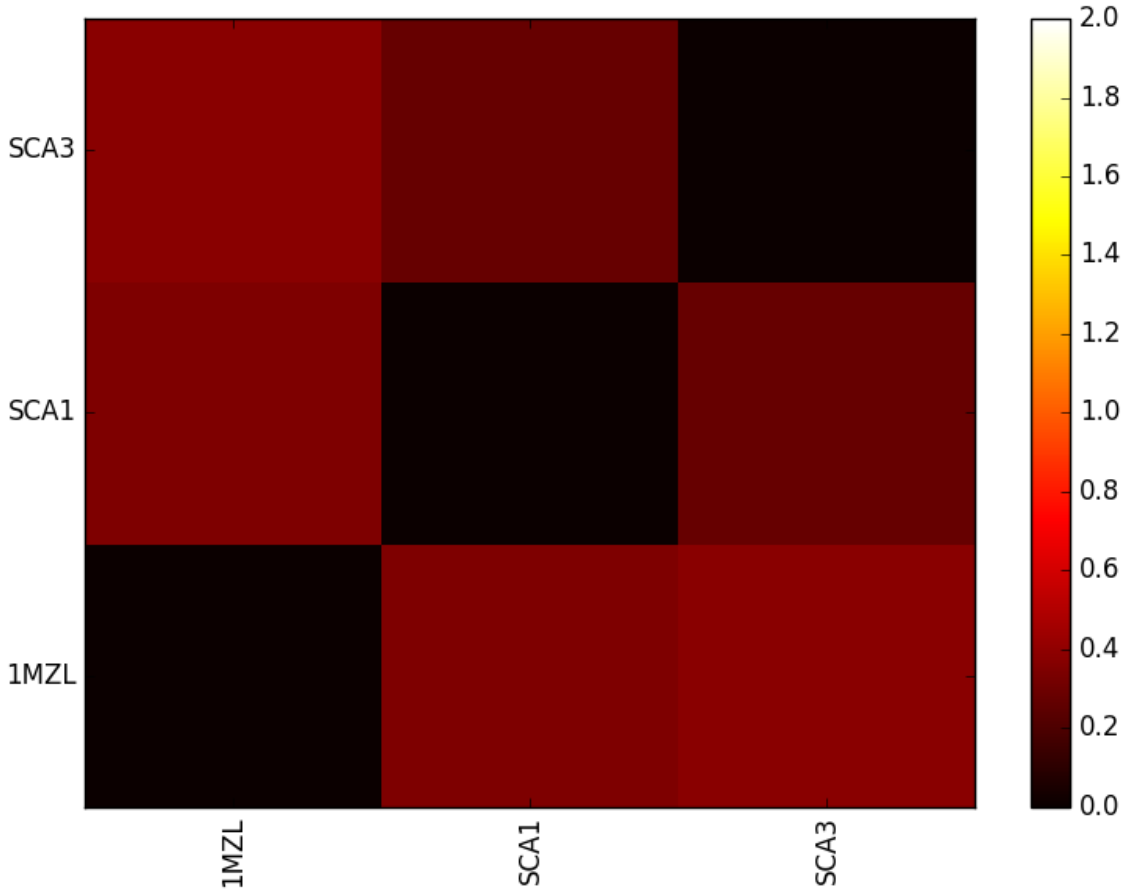
```
plotDend(family, filename='dend.png')
plotESD(family, filename='esd.png')
```

plotDend should produce a dendrogram similar to the following figure.



Proteins that cluster together at lower ESD in the dendrogram are expected to be electrostatically similar.

plotESD should produce a heatmap similar to the following figure.



This heatmap compares all protein pairs in terms of ESD. Lower values once again indicate electrostatic similarity.

If you would like to utilize the interactive plotting function `plotESD_interactive` which generates a heatmap and dendrogram, refer to this notebook demonstration.

If you prefer to export the raw data, you can access the ESD matrix:

```
data = family.esd
```

Other modules such as numpy (example below) or pandas will allow exporting of the ESD matrix to file:

```
import numpy as np
np.savetxt('esd_matrix.txt', data, fmt='%.4f')
```

The ElecSimilarity class also supports calculation of the electrostatic similarity index (ESI) by comparing potentials across all protein structures at corresponding grid points. If you previously calculated the ESD, then you need type the following to perform this calculation:

```
family.calcESI()
```

If you prefer to calculate ESI instead of ESD, you may do so at the time you run the analysis:

```
family.run(esi=True, esd=False, superpose=True)
family.run_parallel(esi=True, esd=False, superpose=True)
```

After performing the ESI calculation, you may view the ESI values by loading the DX file that is located within the

"esi_files" folder of the job directory.

Example case 2: Alascan of a LTP plant protein

AESOP is additionally capapable of comparing electrostatic potentials of alanine mutants for a protein structure. The process follows many of the same steps from example case 1:

```
from aesop import ElecSimilarity, plotDend, plotESD

path_apbs    = 'C:\\APBS\\apbs.exe'
path_pdb2pqr = 'C:\\PDB2PQR\\pdb2pqr.exe'
pdbfiles     = ['1MZL.pdb']
jobname      = 'LTP_test2'

family = ElecSimilarity(pdbfiles=pdbfiles, pdb2pqr_exe=path_pdb2pqr, apbs_exe=path_
↪apbs, jobname=jobname)
```

The analysis can be run with the same `run` or `run_parallel` methods as before; however, AESOP will only know to generate a family of alanine mutants if the length of the `pdbfiles` is 1 or if `selstr` is specified in the run statement. `selstr` will tell AESOP to mutate all ionizable amino acids selected by any of the elements of the list. Only one amino acid will be mutated at a time. The following is an example run statement that will generate mutants for all ionizable amino acids in the 1MZL pdb file:

```
family.run(superpose=False, esd=True, esi=True, selstr=['protein'])
```

References

2.4.6 Alanine Scan

Given a PDB structure of atomistic resolution, the alanine scan method iteratively perturbs the native structure by mutating single amino acids to alanine one residue at a time. In this manner, the method can predict those mutations that are predicted to significantly affect the free energy of association for a complex according to the thermodynamic cycle. For reference, please refer to *[Kieslich2011-2]* and *[Gorham2011-2]* as they present published results for the Barnase-Barstar test system. You may download all necessary files for this example at this link: `download`.

Alascan(pdb[, pdb2pqr_exe, apbs_exe, ...])	Summary Summary of internal varialbles in the Alascan class.

Example case: Barnase-Barstar

Open a new python session, import the Alascan class, and import the plotScan function:

```
from aesop import Alascan, plotScan, writePDB
try:
    from aesop import plotNetwork
except:
    print 'Unable to import plotNetwork, is the NetworkX library installed?'
```

Warning: If you are planning to leverage multiple CPU threads for a faster analysis, please know that extra steps may be required. Specifically, you must protect the entry point of the program according to multiprocessing documentation. You may do this by putting the following code at the beginning of your Python script:

```
if __name__ == '__main__':
    # place remaining code here and maintain level of indentation
```

This precaution becomes unnecessary if you are running the analysis inside an interactive Python session. In the downloadable zip files, we have already placed this protection in the run script, so that you may run the analysis as follows in your platform's terminal:

```
python run_alascan.py
```

Failure to protect the entry point may result in an infinite loop of process spawning.

Next, you must specify the full paths to your `apbs`, `coulomb`, and `pdb2pqr` executables, if the paths for the directories containing the executables have not already been added to the environment. Here is an example for a Windows system:

```
path_apbs    = 'C:\\APBS\\apbs.exe'
path_coulomb = 'C:\\APBS\\coulomb.exe'
path_pdb2pqr = 'C:\\PDB2PQR\\pdb2pqr.exe'
```

Next we will specify the jobname and pdbfile to used in the method. After running the alanine scan, jobname will be used to create a folder where files for the method will be generated. You can download the PDB file for this example from this link (`download`). Make sure you place the PDB in your working directory:

```
jobname = 'alascan'
pdbfile = 'barnase_barstar.pdb'
```

Warning: If you are using your own PDB, make sure the PDB contains no missing heavy atoms. Consider also removing non-standard amino acids. PDBFixer is one option for cleaning PDB files in preparation for AESOP.

The Alascan class will need to know how to define each subunit of the protein complex. To do this, the user should specify a list of selection strings. Each element of the list should be a stand-alone selection string that fully describes how to select the associated subunit. If the selection string list has only 1 element, then be aware that you may only calculate solvation free energies as no association of subunits occurs. In this example, barnase is chain A, and barstar is chain B. Thus, we specify the selection string in the following manner:

```
selstr = ['chain A', 'chain B']
```

Finally, we may initialize the Alanine scan class:

```
alascan = Alascan(pdb=pdbfile,
                  pdb2pqr_exe=path_pdb2pqr,
                  apbs_exe=path_apbs,
                  coulomb_exe=path_coulomb,
                  jobname=jobname,
                  selstr=selstr,
                  minim=False)
```

Note that by default the Alanine scan class will not minimize the structure of mutants. Since the Alascan class seeks to quantify the electrostatic contribution of each amino acid, minimization is unnecessary for our purposes. No clashes should be introduced by the side-chain truncation mutation scheme. If you still prefer to perform minimization, please set minim=True when the class is initialized. In either case, results with or without minimization should be extremely similar.

Alternatively, if paths to apbs, coulomb and pdb2pqr are already added to environment then you may initialize as follows:

```
alascan = Alascan(pdb=pdbfile, jobname=jobname, selstr=selstr)
```

After initialization, you can run the analysis in series:

```
alascan.run()
```

. . . or you can run the analysis in parallel on a certain number of threads (don't pass a number if you wish to use half of available threads):

```
alascan.run_parallel(6)
```

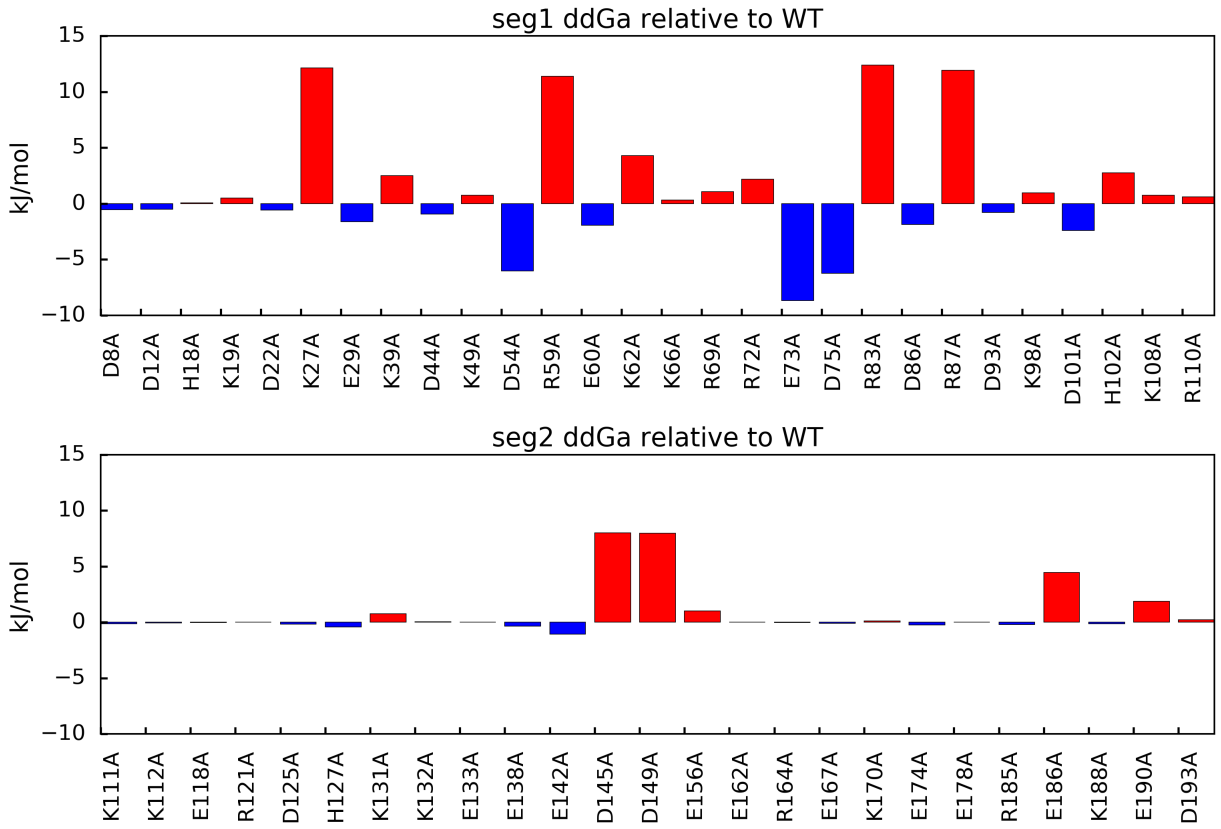
After the run is complete, AESOP will report if any Warnings or Errors were detected in APBS or PDB2PQR. The full logs are stored in the alascan.logs and can be viewed or written to file in the following manner:

```
alascan.viewLogs()
alascan.writeLogs(filename="alascan_logs.txt")
```

Once complete, you can view the results as a barplot:

```
plotScan(alascan, filename='alascan.png')
```

You should end up with a figure similar to the following image:



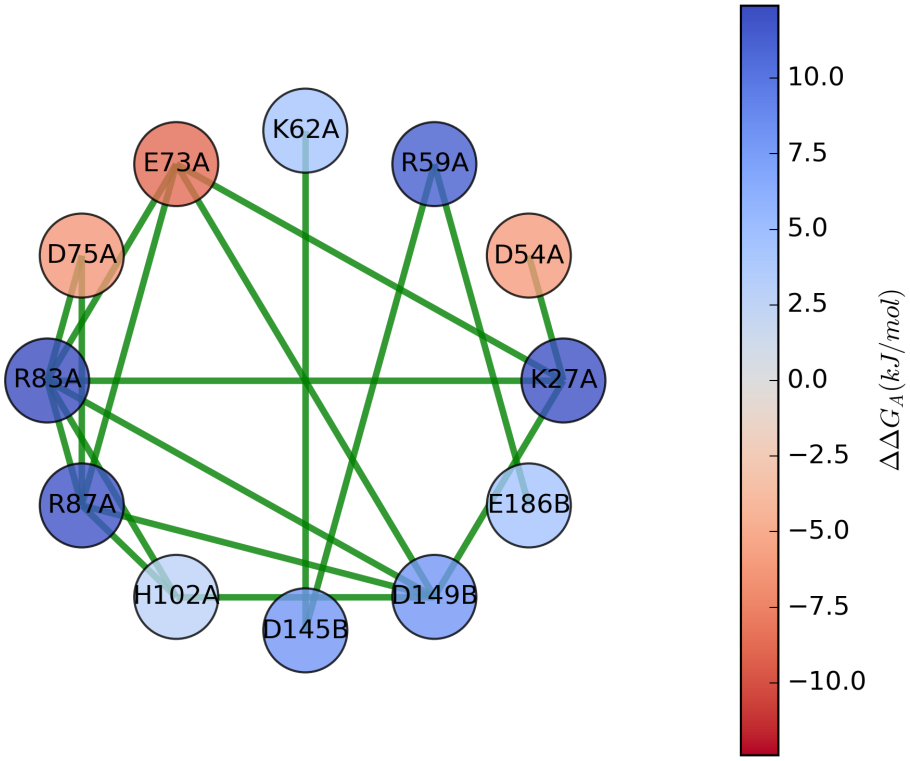
In this figure, mutations that result in positive ddGa values relative to the parent structure suggest loss of binding mutations. This outcome indicates the mutated amino acid was involved in an overall favorable network of electrostatic

interactions in the parent structure. Mutations that result in negative ddGa values relative to the parent structure suggest gain of binding mutations. This outcome indicates the mutated amino acid was involved in an overall unfavorable network of electrostatic interactions in the parent structure.

Alternatively, you can view the results as a network if you have installed the NetworkX library:

```
try:
    plotNetwork(alascan, filename='network.png')
except:
    print 'Skipping plotNetwork example!'
```

You can ignore the try/except blocks with general use. These are here only for those users who do not wish to install Networkx. The network should look similar to the following with standard parameters:



In this figure, nodes are amino acid with significant energy effects. That is, when the residue is mutated the change in free energy of association relative to the parent structure is outside thermal effects (greater than 2.5 kJ/mol or less than -2.5 kJ/mol). Colors of each node are scaled according to the value of this energy, and edges are drawn between Coulombic interactions that occur within some distance cutoff (5 angstroms by default). Node labels consist of the one-letter amino acid code, the residue number, and the chain where the residue is located, in that order. For instance, `R83A` is arginine 83 on chain A.

If you would like to utilize the interactive plotting functions `plotScan_interactive` and `plotNetwork_interactive`, refer to this notebook demonstration.

You may also extract the free energies of association and the associated mutation ids:

```
mut_ids  = alascan.getMutids()
energies = alascan.ddGa_rel()
```

If you wish, you can use build in function to summarize results. If the file name is not specified for the summary, then the summary is simply printed to STDOUT:

```
alascan.summary(filename='alascan_summary.txt')
```

Finally, you may export a PDB file with ddGa values for each residue in the beta-factor column as follows:

```
writePDB(alascan, filename='alascan.ddGa.pdb')
```

References

2.4.7 Directed Mutagenesis Scan

The directed mutagenesis scan is similar to the alanine scan, except it requires specific information about what mutations to perform. Additionally, this method requires an external python library named Modeller. This dependency facilitates more complicated mutations; however, it also requires the method to perform more calculations than for the side-chain truncation method of the alanine scan. For reference, please refer to *[Kieslich2011-2]* and *[Gorham2011-2]* as they present published results for the Barnase-Barstar test system. You may download all necessary files for this example at this link: `download`.

DirectedMutagenesis(pdb, target, mutation[, ...])	Summary

Example case: Barnase-Barstar

Open a new python session, import the DirectedMutagenesis class, and import the plotScan function:

```
from aesop import DirectedMutagenesis, plotScan, writePDB
```

Warning: If you are planning to leverage multiple CPU threads for a faster analysis, please know that extra steps may be required. Specifically, you must protect the entry point of the program according to multiprocessing documentation. You may do this by putting the following code at the beginning of your Python script:

```
if __name__ == '__main__':
    # place remaining code here and maintain level of indentation
```

This precaution becomes unnecessary if you are running the analysis inside an interactive Python session. In the downloadable zip files, we have already placed this protection in the run script, so that you may run the analysis as follows in your platform's terminal:

```
python run_mutagenesis.py
```

Failure to protect the entry point may result in an infinite loop of process spawning.

Next, you must specify the full paths to your `apbs`, `coulomb`, and `pdb2pqr` executables, if the paths for the directories containing the executables have not already been added to the environment. Here is an example for a Windows system:

```
path_apbs    = 'C:\\APBS\\apbs.exe'
path_coulomb = 'C:\\APBS\\coulomb.exe'
```

```
path_pdb2pqr = 'C:\\PDB2PQR\\pdb2pqr.exe'
```

Now we will specify the jobname and pdbfile to used in the method. After running the directed mutagenesis scan, jobname will be used to create a folder where files for the method will be generated. You can download the PDB file for this example from this link (`download`). Make sure you place the PDB in your working directory:

```
jobname = 'directedscan'
pdbfile = 'barnase_barstar.pdb'
```

Warning: If you are using your own PDB, make sure the PDB contains no missing heavy atoms. Consider also removing non-standard amino acids. PDBFixer is one option for cleaning PDB files in preparation for AESOP.

The DirectedMutagenesis class will need to know how to define each subunit of the protein complex. To do this, the user should specify a list of selection strings. Each element of the list should be a stand-alone selection string that fully describes how to select the associated subunit. If the selection string list has only 1 element, then be aware that you may only calculate solvation free energies as no association of subunits occurs. In this example, barnase is chain A, and barstar is chain B. Thus, we specify the selection string in the following manner:

```
selstr = ['chain A', 'chain B']
```

Next, the DirectedMutagenesis will need to know what residues to mutate and the mutation to perform. To accomplish this, we specify another list of selection strings where each element of the list specifies a single residue from the PDB file:

```
target = ['resnum 27',  'resnum 73',  'resnum 83',  'resnum 87',  # mutations in␣
↪chain A
          'resnum 145', 'resnum 149', 'resnum 164', 'resnum 186'] # mutations in␣
↪chain B
```

For each mutation target, an amino acid must be specified using the associated 3 letter code for the mutation. Remember respective elements in target and mutation are linked:

```
mutation = ['ASP', 'LYS', 'GLU', 'GLU', # mutations in chain A
            'ARG', 'ARG', 'ASP', 'LYS'] # mutations in chain B
```

Finally, we may initialize the DirectedMutagenesis scan class:

```
mutscan = DirectedMutagenesis(pdb=pdbfile,
                              pdb2pqr_exe=path_pdb2pqr,
                              apbs_exe=path_apbs,
                              coulomb_exe=path_coulomb,
                              jobname=jobname,
                              selstr=selstr,
                              target=target,
                              mutation=mutation,
                              minim=True)
```

After initialization, you can run the analysis in series:

```
mutscan.run()
```

. . . or you can run the analysis in parallel on a certain number of threads:

```
mutscan.run_parallel(6)
```

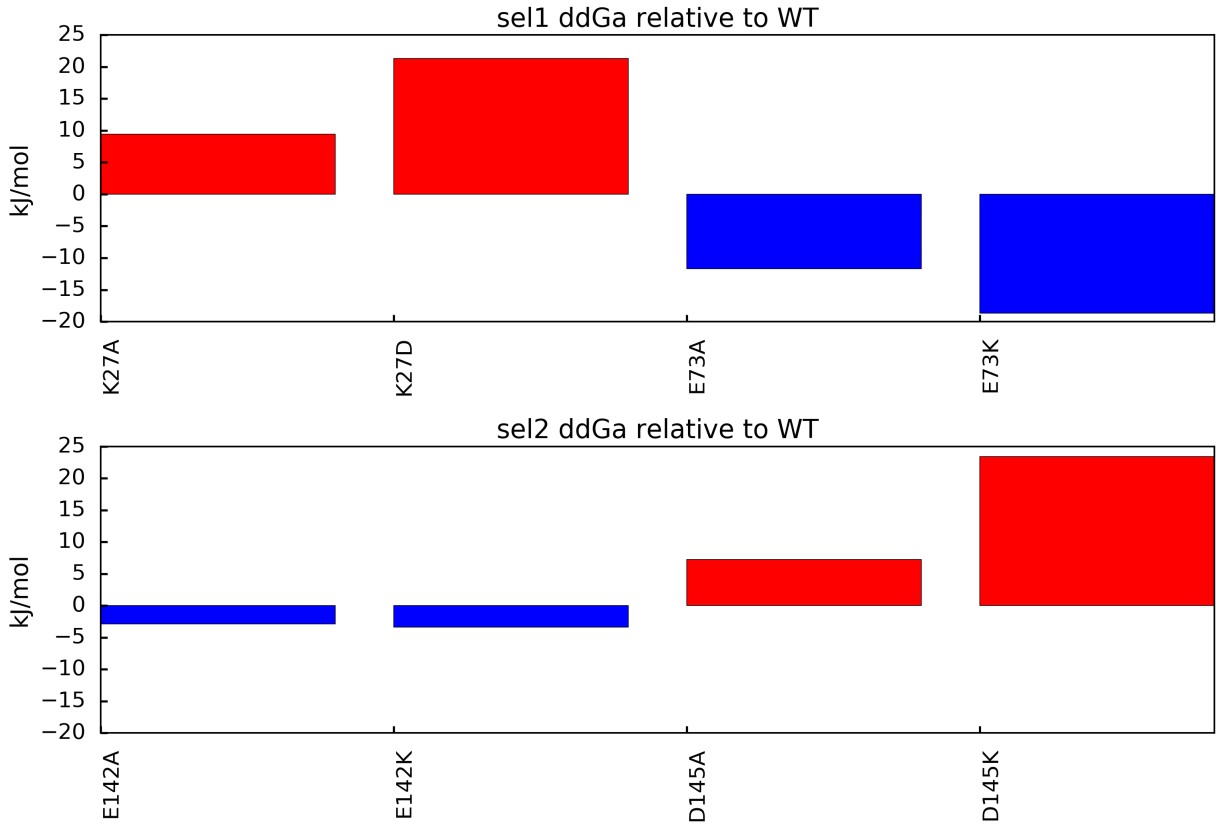
After the run is complete, AESOP will report if any Warnings or Errors were detected in APBS or PDB2PQR. The full logs are stored in the mutscan.logs and can be viewed or written to file in the following manner:

```
mutscan.viewLogs()
mutscan.writeLogs(filename="mutscan_logs.txt")
```

Once complete, you can view the results as a barplot:

```
plotScan(mutscan, filename='directedmutagenesis.png')
```

You should end up with a figure similar to the following image:



In this figure, mutations that result in positive ddGa values relative to the parent structure suggest loss of binding mutations. This outcome indicates the mutated amino acid was involved in an overall favorable network of electrostatic interactions in the parent structure. Mutations that result in negative ddGa values relative to the parent structure suggest gain of binding mutations. This outcome indicates the mutated amino acid was involved in an overall unfavorable network of electrostatic interactions in the parent structure.

If you would like to utilize the interactive plotting functions `plotScan_interactive` and `plotNetwork_interactive`, refer to this notebook demonstration.

You may also extract the free energies of association and the associated mutation ids:

```
mut_ids  = mutscan.getMutids()
energies = mutscan.ddGa_rel()
```

If you wish, you can use build in function to summarize results. If the file name is not specified for the summary, then the summary is simply printed to STDOUT:

```
mutscan.summary(filename='mutscan_summary.txt')
```

Finally, you may export a PDB file with ddGa values for each residue in the beta-factor column as follows:

```
writePDB(mutscan, filename='mutscan.ddGa.pdb')
```

References

2.4.8 API

Package

The aesop submodule contains all relevant classes and functions for the AESOP package.

AESOP contents

exception aesop.aesop.**APBS_Exception**
Bases: exceptions.Exception

class aesop.aesop.**Alascan**(*pdb, pdb2pqr_exe='pdb2pqr', apbs_exe='apbs', coulomb_exe='coulomb', selstr=['protein'], jobname=None, region=None, grid=1, ion=0.15, pdie=20.0, sdie=78.54, ff='parse', cfac=1.5, dx=False, minim=False*)
Summary Summary of internal varialbles in the Alascan class. All parameters are set in the Alascan.__init__(...) and the analysis is run with Alascan.run().

Attributes

apbs [str] Full path to APBS executable. Must be compatible with OS.

apbs_results [str] Full path to output from APBS

cfac [float] Scaling factor for grid dimensions. We suggest to leave this unchanged.

coulomb [str] Full path to coulomb executable from APBS package. Must be compatible with OS.

coulomb_results [str] Full path to folder containing results from coulomb.

dime [list] List of three integers. Parameter required for APBS. Please see description at: http://www.poissonboltzmann.org/docs/apbs-overview/

disu [bool, optional] If True, Modeller will guess the patches for disulfide bridges within the provided protein structures. Only relevant if minim is set to Trueself.

dx [bool] Variable that specifies if potential files should be written to disk. Default is False.

dx_files [list] If written to disk, list of potential files written to disk. The folder containing these files is given by Alascan.apbs_results.

E_ref [ndarray] Description

E_solv [ndarray] Description

ff [str] Force-field to use for PDB2PQR

file_pdb_template [TYPE] Description

gcent [list] List of three integers. Parameter required for APBS. Please see description at: http://www.poissonboltzmann.org/docs/apbs-overview/

Gcoul [ndarray] Coulombic free energies, corresponding to Alascan.mutid.

glen [list] List of three integers. Parameter required for APBS. Please see description at: http://www.poissonboltzmann.org/docs/apbs-overview/

Gref [ndarray] Reference-state free energies, corresponding to Alascan.mutid.

grid [numeric] Distance spacing of grid points. If the grid dimensions are not divisible by three, resolution will be increased (smaller grid spacing) until grid dimensions are divisible by three.

Gsolv [ndarray] Solvated-state free energies, corresponding to Alascan.mutid.

ion [numeric] Ionic strength to be used in the solvated-state APBS calculations.

jobdir [str] [Optional] Path to folder containing results. If not specified, a directory will be generated.

jobname [str] [Optional] Name for current job, will be used to create the jobdir.

list_chids [list] Chain ID where mutation was made. Corresponds to Alascan.mutid.

list_resnames [list] Residue names where mutation was made. Corresponds to Alascan.mutid.

list_resnums [list] Residue numbers where mutation was made. Corresponds to Alascan.mutic.

logs [list] List of strings that represent the log files from every executable called (namely, PDB2PQR and APBS)

logs_apbs_dir [str] Folder in jobdir containing output from APBS (logs, input files, dx files)

mask_by_sel [ndarray] Matrix containing selection masks. The first column corresponds to the selection string for the parent and each column thereafter corresponds to an element of the selection string (selstr) in the same order.

max_iter [integer, optional] If minimization is to be performed, this parameter limits the maximum number of calls to the objective function before minimization is terminated. Default is 1000 iterations.

min_atom_shift [float, optional] If minimization is to be performed, this parameter will determine the convergence criteria. If the maximum atomic shift for all atoms between minimization steps is less than this value, then minimization is terminated. Default value is 0.1 angstroms.

minim [bool, optional] If True, minimization will be performed with Modeller's conjugate gradient descent algorithm. Default is False for the Alanine scan class as no clashes should result from mutations.

mutid [list] List of mutant IDs. The first element corresponds to the parent. Subsequent elements correspond to each element of the selection string list (selstr). Please use Alascan.getMutids() to get vectorized version.

output [string, optional] If minimization is performed, this parameter deterimines what output to STDOUT Modeller will use. 'NO_REPORT' results in a minimal output to STDOUT, while 'REPORT' results in a more verbose output to STDOUT.

pdb [str] Path to PDB file with atomic coordinates. Must follow formatting conventions of the Protein Databank.

pdb2pqr [str] Full path to PDB2PQR executable.

pdb_complex_dir [str] Folder name in the job directory that contains the PDB file(s) of the complex structures.

pdie [numeric] Protein dielectric constant to be used in APBS.

pqr_complex_dir [str] Folder name in the job directory containing PQR files for each protein complex.

pqr_sel_dir [list] List of folder names in the job directory that contain PQR files for selection strings (selstr). Each element of pqr_sel_dir corresponds to the same element of selstr.

region [list] List of additional selection strings specifying the zone where mutations should occur. Generally unused, unless a region of interest is involved. Each element of region should correspond to the same element of selstr. That is, each region selection string will futher narrow down the initial selection string.

sdie [numeric] Solvent dielectric constant to be used in APBS.

selstr [list] List of selection strings. Typically each selection string will correspond to a chain in a protein complex. We advise users to specify two or more selection strings. (Ex: ['chain A', 'chain B'])

Methods

calcAPBS()	Summary Run APBS on each structure in mutant library, in serial.
calcAPBS_parallel([n_workers])	Summary Run APBS on each structure in mutant library, in parallel.
calcCoulomb()	Summary Calculates Coulombic free energies with coulomb.exe from the APBS toolbox.
calcCoulomb_parallel([n_workers])	Summary Calculates Coulombic free energies with coulomb.exe from the APBS toolbox in a parallel manner.
calcESI([idx])	Summary
dGsolv_rel()	Summary Calculates and returns the free energy of a solvation for each mutant relative to the parent free energy of solvation.
ddGa_rel()	Summary Calculates and returns the free energy of association relative to the parent free energy of association.
find_grid()	Summary
genDirs()	Summary This subroutine will generate all directories needed to contain structural files, logs, etc.
genMutid()	Summary This subroutine reads the input PDB, selects the structure where mutations will occur, and saves all mutant IDs in the class.
genParent()	Summary Reads PDB file specified in the constructor; applies and combines results from the selection strings; and saves the final template structure in the job directory.
genTruncatedPQR()	Summary Generate a structure for each mutant ID by truncating the side chain to form alanine.
getMutids()	Summary
run()	Summary Perform a compuational alanine scan on the initialized Alascan class.

Continued on next page

Table 2.4 – continued from previous page

run_parallel([n_workers])	Summary Perform a computational alanine scan on the initialized Alascan class using multiple processes in parallel.
set_grid(dime, glen, gcent)	Summary In the case that the user wishes to manually specify grid dimension, this may be accomplished with the set_grid method.
summary([filename])	Summary Summarize results from the computational alanine scan once complete.

checkerrors	
checkwarnings	
viewLogs	
writeLogs	

calcAPBS()
Summary Run APBS on each structure in mutant library, in serial.

Returns

None Stores energies from APBS in Gref and Gsolv class attributes.

calcAPBS_parallel(*n_workers=None*)
Summary Run APBS on each structure in mutant library, in parallel.

Parameters

n_workers [int] Number of processes to run. If None, method will use all available threads.

Returns

None Saves solvated-state and reference-state free energies as class attributes.

calcCoulomb()
Summary Calculates Coulombic free energies with coulomb.exe from the APBS toolbox.

Returns

None Saves Coulombic free energies as a class attribute.

calcCoulomb_parallel(*n_workers=None*)
Summary Calculates Coulombic free energies with coulomb.exe from the APBS toolbox in a parallel manner.

Parameters

n_workers [int] Number of processes to run. If None, method will use all available threads.

Returns

None Saves Coulombic free energies as a class attribute.

calcESI(*idx=-1*)
Summary

Compare potential files and calculate the similarity index. Values closer to 1 imply similarity while values closer to zero imply dissimilarity.

Parameters

method [str, optional] This parameter will allow for other metrics to compare grid potentials; however, for now only 'AND' is implemented.

idx [int] Index of original PDB files supplied containing reference structure. Set to None to perform all pairwise comparisons.

Returns

None Writes esi files to the esi_files directory within the job directory.

checkerrors()

checkwarnings()

dGsolv_rel()

Summary Calculates and returns the free energy of a solvation for each mutant relative to the parent free energy of solvation.

Returns

ndarray Array of solvation free energies corresponding to mutant IDs from the Alascan.getMutIDs() method.

ddGa_rel()

Summary Calculates and returns the free energy of association relative to the parent free energy of association.

Returns

ndarray Array of free energies corresponding to the mutant IDs from the Alascan.getMutIDs() method.

find_grid()

Summary Calculate grid dimensions for APBS (dime, glen, gcent)

Returns

TYPE Sets class attributes dime, glen, and gcent

genDirs()

Summary This subroutine will generate all directories needed to contain structural files, logs, etc. In the future we may implement a method to remove such files when outputs are more standardized.

Returns

None No output, operates on the class and generates folders in the job directory.

genMutid()

Summary This subroutine reads the input PDB, selects the structure where mutations will occur, and saves all mutant IDs in the class. If region is specified in the constructor, then the constraint will be applied here.

Returns

None Operates on the class to generate a list of mutant IDs for each selection string. The following class variables will be generated (see class description):

- mutid
- list_chids
- list_resnums
- list_resnames
- mask_by_sel

genParent()

Summary Reads PDB file specified in the constructor; applies and combines results from the selection strings; and saves the final template structure in the job directory.

Returns

None Template pdb written in job directory and location saved in Alascan.file_pdb_template.

genTruncatedPQR()
Summary Generate a structure for each mutant ID by truncating the side chain to form alanine.

Returns

None Write library of mutant structures to disk for subsequent analysis.

getMutids()
Summary

Returns

list Returns vectorized format of mutids.

run()
Summary Perform a compuational alanine scan on the initialized Alascan class.

Returns

None Outputs text to STDOUT when run is complete, will be made optional in the future.

run_parallel(*n_workers=None*)
Summary Perform a computational alanine scan on the initialized Alascan class using multiple processes in parallel.

Parameters

n_workers [int] Number of processes to run. If None, method will use all available threads.

Returns

None Outputs text to STDOUT when run is complete, will be made optional in the future.

set_grid(*dime*, *glen*, *gcent*)
Summary In the case that the user wishes to manually specify grid dimension, this may be accomplished with the set_grid method. Typically, this is used when grid space parameters must be consistent for many analyses. Please see description at: http://www.poissonboltzmann.org/docs/apbs-overview/ for description of parameters.

Parameters

dime [list] List of three integers.

glen [list] List of three integers.

gcent [list] List of three integers.

Returns

TYPE Description

summary(*filename=None*)
Summary Summarize results from the computational alanine scan once complete.

Parameters

filename [str, optional] In order to write summary to a text file, supply the filename (full path).

Returns

None Prints summary of residues and energies relative to the parent structure if no filename is provided. Otherwise, writes to text file.

viewLogs()

writeLogs(*filename=None*)

exception aesop.aesop.**Complete_PDB_Exception**
Bases: exceptions.Exception

class aesop.aesop.**DirectedMutagenesis**(*pdb, target, mutation, pdb2pqr_exe='pdb2pqr', apbs_exe='apbs', coulomb_exe='coulomb', selstr=['protein'], jobname=None, grid=1, ion=0.15, pdie=20.0, sdie=78.54, ff='parse', cfac=1.5, dx=False, minim=True*)

Summary

Attributes

apbs [str] Full path to APBS executable. Must be compatible with OS.

apbs_results [str] Full path to output from APBS

cfac [numeric] Scaling factor for grid dimensions. We suggest to leave this unchanged.

coulomb [str] Full path to coulomb executable from APBS package. Must be compatible with OS.

coulomb_results [str] Full path to folder containing results from coulomb.

dime [list] List of three integers. Parameter required for APBS. Please see description at: http://www.poissonboltzmann.org/docs/apbs-overview/

disu [bool, optional] If true, Modeller will guess patches for disulfide bridges. Only relevant if minim is set to True.

dx [bool] Variable that specifies if potential files should be written to disk. Default is False.

dx_files [list] If written to disk, list of potential files written to disk. The folder containing these files is given by Alascan.apbs_results.

E_ref [ndarray] Reference state energy values for each structure from APBS.

E_solv [ndarray] Solvated state energy values for each structure from APBS.

ff [string] Force-field to use for PDB2PQR

file_pdb_template [string] Full path to template PDB file.

gcent [list] List of three integers. Parameter required for APBS. Please see description at: http://www.poissonboltzmann.org/docs/apbs-overview/

Gcoul [ndarray] Coulombic free energies, corresponding to Alascan.mutid.

glen [list] List of three integers. Parameter required for APBS. Please see description at: http://www.poissonboltzmann.org/docs/apbs-overview/

Gref [ndarray] Reference-state free energies, corresponding to Alascan.mutid.

grid [numeric] Distance spacing of grid points. If the grid dimensions are not divisible by three, resolution will be increased (smaller grid spacing) until grid dimensions are divisible by three.

Gsolv [ndarray] Solvated-state free energies, corresponding to Alascan.mutid.

ion [numeric] Ionic strength to be used in the solvated-state APBS calculations.

jobdir [str] [Optional] Path to folder containing results. If not specified, a directory will be generated.

jobname [str] [Optional] Name for current job, will be used to create the jobdir.

list_chids [list] Chain ID where mutation was made. Corresponds to Alascan.mutid.

list_resnames [list] Residue names where mutation was made. Corresponds to Alascan.mutid.

list_resnums [list] Residue numbers where mutation was made. Corresponds to Alascan.mutic.

logs [list] List of strings that represent the log files from every executable called (namely, PDB2PQR and APBS)

logs_apbs_dir [str] Folder in jobdir containing output from APBS (logs, input files, dx files)

mask_by_sel [ndarray] Matrix containing selection masks. The first column corresponds to the selection string for the parent and each column thereafter corresponds to an element of the selection string (selstr) in the same order.

max_iter [integer, optional] Maximum number of calls to the objective function. If this value is reached, then minimization is terminated. Default value is 1000 iterations.

min_atom_shift [float, optional] If the maximimum atomic shift between minimization steps is less thant this value, convergence is reached and minimization is terminated. Default value is 0.1 angstroms.

minim [bool, optional] If true, structures will be minimzed with Modeller's conjugate gradient descent algorithm.

mutation [list] Identity of amino acid for mutation of corresponding target. Must be the same length as target and each residue must use the standard 3-letter amino acid code.

mutid [list] List of mutant IDs. The first element corresponds to the parent. Subsequent elements correspond to each element of the selection string list (selstr). Please use Alascan.getMutids() to get vectorized version.

output [string, optional] Modeller option specifying whether to print verbose output to STDOUT ('REPORT'), or to print minimal output to STDOUT ('NO_ REPORT')

pdb [str] Path to PDB file with atomic coordinates. Must follow formatting conventions of the Protein Databank.

pdb2pqr [str] Full path to PDB2PQR executable.

pdb_complex_dir [str] Folder name in the job directory that contains the PDB file(s) of the complex structures.

pdie [numeric] Protein dielectric constant to be used in APBS.

pqr_complex_dir [str] Folder name in the job directory containing PQR files for each protein complex.

pqr_sel_dir [list] List of folder names in the job directory that contain PQR files for selection strings (selstr). Each element of pqr_sel_dir corresponds to the same element of selstr.

sdie [numeric] Solvent dielectric constant to be used in APBS.

selstr [list] List of selection strings. Typically each selection string will correspond to a chain in a protein complex. We advise users to specify two or more selection strings. (Ex: ['chain A', 'chain B'])

target [list] List of residue numbers to mutate. Must correspond element-wise to mutation attribute.

Methods

calcAPBS()	Summary Call apbs to calculate reference-state and solvation-state energies for all structures in library.
calcAPBS_parallel([n_workers])	Summary Run APBS on each structure in mutant library, in parallel.
calcCoulomb()	Summary Calculates Coulombic free energies with coulomb.exe from the APBS toolbox.
calcCoulomb_parallel([n_workers])	Summary Calculates Coulombic free energies with coulomb.exe from the APBS toolbox in a parallel manner.
dGsolv_rel()	Summary Calculates and returns the free energy of a solvation for each mutant relative to the parent free energy of solvation.
ddGa_rel()	Summary Calculates and returns the free energy of association relative to the parent free energy of association.
find_grid()	Summary
genDirs()	Summary This subroutine will generate all directories needed to contain structural files, logs, etc.
genMutid()	Summary This subroutine reads the input PDB, selects the structure where mutations will occur, and saves all mutant IDs in the class.
genPDB([minim])	Summary
genPQR()	Summary Generates PQR for each PDB in library.
genParent()	Summary Reads PDB file specified in the constructor; applies and combines results from the selection strings; and saves the final template structure in the job directory.
getMutids()	Summary
run()	Summary Perform a directed mutagenesis scan on the initialized class.
run_parallel([n_workers])	Summary Perform a computational directed mutagenesis scan on the initialized class using multiple processes in parallel.
set_grid(dime, glen, gcent)	Summary In the case that the user wishes to manually specify grid dimension, this may be accomplished with the set_grid method.
summary([filename])	Summary Summarize results from the computational alanine scan once complete.

checkerrors	
checkwarnings	
viewLogs	
writeLogs	

calcAPBS()

Summary Call apbs to calculate reference-state and solvation-state energies for all structures in library.

Returns

None Sets class attributes Gsolv and Gref

calcAPBS_parallel(*n_workers=None*)
Summary Run APBS on each structure in mutant library, in parallel.

Parameters

n_workers [int] Number of processes to run. If None, method will use all available threads.

Returns

None Saves solvated-state and reference-state free energies as class attributes.

calcCoulomb()
Summary Calculates Coulombic free energies with coulomb.exe from the APBS toolbox.

Returns

None Saves Coulombic free energies as a class attribute.

calcCoulomb_parallel(*n_workers=None*)
Summary Calculates Coulombic free energies with coulomb.exe from the APBS toolbox in a parallel manner.

Parameters

n_workers [int] Number of processes to run. If None, method will use all available threads.

Returns

None Saves Coulombic free energies as a class attribute.

checkerrors()

checkwarnings()

dGsolv_rel()
Summary Calculates and returns the free energy of a solvation for each mutant relative to the parent free energy of solvation.

Returns

ndarray Array of solvation free energies corresponding to mutant IDs from the Alascan.getMutIDs() method.

ddGa_rel()
Summary Calculates and returns the free energy of association relative to the parent free energy of association.

Returns

ndarray Array of free energies corresponding to the mutant IDs from the Alascan.getMutIDs() method.

find_grid()
Summary Calculate grid dimensions for APBS (dime, glen, gcent)

Returns

TYPE Sets class attributes dime, glen, and gcent

genDirs()
Summary This subroutine will generate all directories needed to contain structural files, logs, etc. In the future we may implement a method to remove such files when outputs are more standardized.

Returns

None No output, operates on the class and generates folders in the job directory.

genMutid()
Summary This subroutine reads the input PDB, selects the structure where mutations will occur, and saves all mutant IDs in the class. If region is specified in the constructor, then the constraint will be applied here.

Returns

None Operates on the class to generate a list of mutant IDs for each selection string. The following class variables will be generated (see class description):

- mutid
- list_chids
- list_resnums
- list_resnames
- mask_by_sel

genPDB(*minim=True*)
Summary Generates mutations by calling function to mutate PDB with modeller

Returns

None Write PDB library to expected path according to class attributes.

genPQR()
Summary Generates PQR for each PDB in library.

Returns

None Calls PDB2PQR and writes PQR to expected path according to class attributes.

genParent()
Summary Reads PDB file specified in the constructor; applies and combines results from the selection strings; and saves the final template structure in the job directory.

Returns

None Template pdb written in job directory and location saved in Alascan.file_pdb_template.

getMutids()
Summary

Returns

list Returns vectorized format of mutids.

run()
Summary Perform a directed mutagenesis scan on the initialized class.

Returns

None Outputs text to STDOUT when run is complete, will be made optional in the future.

run_parallel(*n_workers=None*)
Summary Perform a computational directed mutagenesis scan on the initialized class using multiple processes in parallel.

Parameters

n_workers [int] Number of processes to run. If None, method will use all available threads.

Returns

None Outputs text to STDOUT when run is complete, will be made optional in the future.

set_grid(*dime*, *glen*, *gcent*)

Summary In the case that the user wishes to manually specify grid dimension, this may be accomplished with the set_grid method. Typically, this is used when grid space parameters must be consistent for many analyses. Please see description at: http://www.poissonboltzmann.org/docs/apbs-overview/ for description of parameters.

Parameters

dime [list] List of three integers.

glen [list] List of three integers.

gcent [list] List of three integers.

Returns

TYPE Sets class attributes for dime, glen, gcent

summary(*filename=None*)

Summary Summarize results from the computational alanine scan once complete.

Parameters

filename [str, optional] In order to write summary to a text file, supply the filename (full path).

Returns

None Prints summary of residues and energies relative to the parent structure if no filename is provided. Otherwise, writes to text file.

viewLogs()

writeLogs(*filename=None*)

class aesop.aesop.**ElecSimilarity**(*pdbfiles*, *pdb2pqr_exe='pdb2pqr'*, *apbs_exe='apbs'*, *selstr=None*, *jobname=None*, *grid=1*, *ion=0.15*, *pdie=20.0*, *sdie=78.54*, *ff='parse'*, *cfac=1.5*, *minim=False*)

Summary

Attributes

apbs [str] Full path to APBS executable. Must be compatible with OS.

cfac [numeric] Scaling factor for grid dimensions. We suggest to leave this unchanged.

dim_dx [list] Dimensions of grid space

dime [list] List of three integers. Parameter required for APBS.Please see description at: http://www.poissonboltzmann.org/docs/apbs-overview/

disu [bool, optional] If True, Modeller will guess the patches for disulfide bridges within the provided protein structures. Only relevant if minim is set to True.

dx [bool] Variable that specifies if potential files should be written to disk. Default is False.

dxdir [str] Folder in job directory where potential files are stored.

dxfiles [list] List of all potential files.

edges [ndarray] Edges of grid space.

esd [ndarray] Matrix of pairwise electrostatic similarities.

ff [str] Forcefield to use in assigning charges to PDB files.

gcent [list] List of three integers. Parameter required for APBS. Please see description at: http://www.poissonboltzmann.org/docs/apbs-overview/

glen [list] List of three integers. Parameter required for APBS. Please see description at: http://www.poissonboltzmann.org/docs/apbs-overview/

grid [int] Desired grid spacing in Angstroms. Actual grid spacing may be slightly lower.

ids [list] List of IDs for each structure being compared with the ESD metric.

ion [numeric] Ionic strength to be used in the solvated-state APBS calculations.

jobdir [TYPE] Description

jobname [str] [Optional] Name for current job, will be used to create the jobdir.

logs [list] List of strings that represent the log files from every executable called (namely, PDB2PQR and APBS)

max_iter [integer, optional] Maximum number of calls to the objective function. If this value is reached, then minimization is terminated. Default value is 1000 iterations.

midpoints [ndarray] Midpoints of grid space

min_atom_shift [float, optional] If the maximimum atomic shift between minimization steps is less thant this value, convergence is reached and minimization is terminated. Default value is 0.1 angstroms.

minim [bool, optional] If true, structures will be minimzed with Modeller's conjugate gradient descent algorithm.

output [string, optional] Modeller option specifying whether to print verbose output to STDOUT ('REPORT'), or to print minimal output to STDOUT ('NO_ REPORT')

pdb2pqr [str] Full path to PDB2PQR executable.

pdbdir [str] Folder in job directory where PDB files are located.

pdbfiles [list] List of PDB file names.

pdie [numeric] Protein dielectric constant to be used in APBS.

pqrdir [str] Folder in job directory containing PQR files.

pqrfiles [list] List of PQR files in pqrdir

sdie [numeric] Solvent dielectric constant to be used in APBS.

Methods

calcESD([method])	Summary Compare potential files and calculate the similarity distance.
calcESI([method, idx])	Summary
centerPDB()	Summary
genDX()	Summary Generates potential files using APBS.
genDX_parallel([n_workers])	Summary Generates multiple potential files in parallel by calling APBS multiple times according to how many threads are available/specified.
genPQR()	Summary
initializeGrid()	Summary Method to find grid parameters and ensure that the product of dimensions is divisible by three.
minimPDB()	Summary Re-reads PDB files in pdbdir and performs energy minimization.

Continued on next page

Table 2.6 – continued from previous page

mutatePQR([selstr, minim, ff])	Summary
superposePDB()	Summary Superpose each structure in pdbfiles with first element in pdbfiles list.

checkerrors	
checkwarnings	
run	
run_parallel	
viewLogs	
writeLogs	

calcESD(*method='AND'*)

Summary Compare potential files and calculate the similarity distance. Smaller distances imply similarity.

Parameters

method [str, optional] This parameter will allow for other metrics to compare grid potentials; however, for now only 'AND' is implemented.

Returns

None Stores esd matrix as class attribute.

calcESI(*method='AND'*, *idx=0*)

Summary

Compare potential files and calculate the similarity index. Values closer to 1 imply similarity while values closer to zero imply dissimilarity.

Parameters

method [str, optional] This parameter will allow for other metrics to compare grid potentials; however, for now only 'AND' is implemented.

idx [int] Index of original PDB files supplied containing reference structure. Set to None to perform all pairwise comparisons.

Returns

None Writes esi files to the esi_files directory within the job directory.

centerPDB()

Summary Re-reads PDB file in pdbdir and centers coordinates at (0, 0, 0)

Returns

TYPE Overwrites previous PDB files in pdbdir

checkerrors()

checkwarnings()

genDX()

Summary Generates potential files using APBS.

Returns

None Generates DX files in dxdir

genDX_parallel(*n_workers=None*)

Summary Generates multiple potential files in parallel by calling APBS multiple times according to how many threads are available/specified.

Parameters

n_workers [int] Number of processes to run. If None, method will use all available threads.

Returns

TYPE Generates DX files in dxdir.

genPQR()
Summary Convert all PDB files to PQR files with charges allocated according to a compatible force-field

Returns

None Generates PQR files in the pqrdir

initializeGrid()
Summary Method to find grid parameters and ensure that the product of dimensions is divisible by three.

Returns

None Sets class attributes dime, glen, gcent.

minimPDB()
Summary Re-reads PDB files in pdbdir and performs energy minimization.

Returns

TYPE Overwrites previous PDB files in pdbdir

mutatePQR(*selstr=['protein'], minim=False, ff='parse'*)
Summary Mutate all PQR files, optional method

Returns

None Generates PQR files in the pqrdir

run(*center=False*, *superpose=False*, *esi=False*, *esd=True*, *selstr=None*, *idx=0*, *minim=False*)

run_parallel(*n_workers=None*, *center=False*, *superpose=False*, *esi=False*, *esd=True*, *selstr=None*, *idx=0*, *minim=False*)

superposePDB()
Summary Superpose each structure in pdbfiles with first element in pdbfiles list. This uses Modeller to perform the superpositioning.

Returns

TYPE Overwrites PDB files in pdbdir with new coordinate information.

viewLogs()

writeLogs(*filename=None*)

class aesop.aesop.**Grid**(*filename=None*)
Summary

The grid class facilitates parsing and writing of OpenDX file formats. In the current state, the class is quite rudimentary and only supports changing vectors for the grid data.

Attributes

filename [string] DX file to import

pot [ndarray] Vectors at each grid point. For AESOP, these will typically be electrostatic potentials or an electrostatic similarity index.

header [list] List of grid parameters from the OpenDX format prior to vectors.

footer [list] List of grid parameters from the OpenDX format subsequent to the vectors.

Methods

readDX([filename])	Summary
writeDX([filename])	Summary

re = <module 're' from '/home/docs/checkouts/readthedocs.org/user_builds/aesop/envs/la

readDX(*filename=None*)

Summary

Method to parse a DX file

Parameters

filename [string] Name for the OpenDX file to be imported. If unspecified, this parameter defaults to the class attribute.

writeDX(*filename=None*)

Summary

Function to write OpenDX files

Parameters

filename [string] Name for OpenDX file that will be written. This should be a full path if you wish to place the file somewhere other than the current working directory.

exception aesop.aesop.**Minimize_CG_Exception**

Bases: exceptions.Exception

exception aesop.aesop.**PDB2PQR_Exception**

Bases: exceptions.Exception

aesop.aesop.**batchAPBS**(*kernel*)

Summary Function required to run multiple APBS jobs simultaneously. Not intended for general use.

Parameters

kernel [tuple] Tuple of parameters required for APBS.

Returns

ndarray i, j represent the index in the matrix with which the calculated energies correspond. The last two elements are the solvation and reference energies, respectively.

aesop.aesop.**batchCalcDX**(*kernel*)

Summary Function required to run multiple APBS jobs simultaneously. Not intended for general use.

Parameters

kernel [tuple] Tuple of parameters required for APBS.

Returns

None Writes files according to calcDX function.

aesop.aesop.**batchCoulomb**(*kernel*)

Summary Function required to run multiple Coulomb jobs simultaneously. Not intended for general use.

Parameters

kernel [tuple] Tuple of parameters required for APBS.

Returns

ndarray i, j represent the index in the matrix with which the calculated energies correspond. The last element is the Coulombic energy.

aesop.aesop.**calcDX**(*path_apbs_exe*, *pqrfile*, *prefix=None*, *grid=1.0*, *ion=0.15*, *pdie=20.0*, *sdie=78.54*, *cfac=1.5*, *glen=None*, *gcent=array([0., 0., 0.])*, *dime=None*)

Summary Calls the APBS executable after generating the APBS inputfile. Generates a potential file (DX).

Parameters

path_apbs_exe [str] Full path to APBS executable, EX: 'C:APBSapbs.exe'.

pqrfile [str] The PQR file that will be used to generate a grid of electrostatic potentials. Must be a full path if file is not in current path.

prefix [str, optional] Phrase to prepend before any file that is generated before writing.

grid [float, optional] Distance spacing of grid points. If the grid dimensions are not divisible by three, resolution will be increased (smaller grid spacing) until grid dimensions are divisible by three.

ion [float, optional] Ionic strength for APBS calculation.

pdie [float, optional] Protein dielectric constant for APBS calculation.

sdie [float, optional] Solvent dielectric constant for APBS calculation.

cfac [float, optional] Scaling factor for grid dimensions. We suggest to leave this unchanged.

glen [None, optional] List of three integers. Parameter required for APBS. Please see description at: http://www.poissonboltzmann.org/docs/apbs-overview/

gcent [TYPE, optional] List of three integers. Parameter required for APBS. Please see description at: http://www.poissonboltzmann.org/docs/apbs-overview/

dime [None, optional] List of three integers. Parameter required for APBS. Please see description at: http://www.poissonboltzmann.org/docs/apbs-overview/

Returns

(log, err) [tuple] When APBS runs, outputs that would have been sent to STDOUT are captured. Log contains the run log and err contains errors.

aesop.aesop.**complete_structure**(*pdb*, *dest=None*, *disu=False*)

Summary Function to fill in residues with missing atoms. This method simply calls complete_pdb from Modeller.

Parameters

pdb [str] Full path to pdbfile that will be modified.

dest [str (optional)] Full path to destination where completed pdb will be written. If not specified, the model object from Modeller will be returned.

disu [bool (optional)] If True, complete_pdb will predict and patch all disulfide bridges. Default is False.

aesop.aesop.**execAPBS**(*path_apbs_exe*, *pqr_chain*, *dime*, *glen*, *gcent*, *prefix=None*, *ion=0.15*, *pdie=20.0*, *sdie=78.54*, *dx=False*)

Summary Calls the APBS executable after generating the APBS inputfile. Calculates solvation and reference energies.

Parameters

path_apbs_exe [str] Full path to APBS executable, EX: 'C:APBSapbs.exe'.

pqr_chain [str] PQR file name containing the segment that will undergo electrostatic calculations.

dime [list] List of three integers. Parameter required for APBS. Please see description at: http://www.poissonboltzmann.org/docs/apbs-overview/

glen [list] List of three integers. Parameter required for APBS. Please see description at: http://www.poissonboltzmann.org/docs/apbs-overview/

gcent [list] List of three integers. Parameter required for APBS. Please see description at: http://www.poissonboltzmann.org/docs/apbs-overview/

prefix [str, optional] Phrase to prepend before any file that is generated before writing.

ion [float, optional] Ionic strength for APBS calculation.

pdie [float, optional] Protein dielectric constant for APBS calculation.

sdie [float, optional] Solvent dielectric constant for APBS calculation.

dx [bool, optional] If true, potential files are written.

Returns

file_apbs_log [str] File name for the log file that APBS generates. This file contains results from performed calculations.

aesop.aesop.**execCoulomb**(*path_coulomb_exe*, *pqr*)

Summary Call Coulomb from APBS tools to calculate Coulombic energies.

Parameters

path_coulomb_exe [str] Full path to coulomb executable.

pqr [TYPE] Filename for PQR to use for Coulombic energy calculation. Must be full path if not in current path.

Returns

float Coulombic energy associated with input PQR file.

aesop.aesop.**execPDB2PQR**(*path_pdb2pqr_exe*, *pdbfile*, *outfile=None*, *ff='parse'*)

Summary Calls the APBS executable according to: <path to pdb2pqr appropriate for OS> –ff=parse –chain inputfile outputfile

Parameters

path_pdb2pqr_exe [str] Full path to pdb2pqr executable

pdbfile [str] PDB file to be converted to a PQR. Should be a full path if not in current working directory.

outfile [str, optional] File name for PQR file that will be generated. May be a full path if desired output is not in current working directory.

ff [str, optional] String instructing PDB2PQR what force field to employ. For more information visit: http://www.poissonboltzmann.org/docs/pdb2pqr-usage/

Returns

(log, err) [tuple] When PDB2PQR runs, outputs that would have been sent to STDOUT are captured. Log contains the run log and err contains errors.

aesop.aesop.**minimize_cg**(*struct*, *dest=None*, *disu=True*, *min_atom_shift=0.1*, *max_iter=1000*, *output='NO_REPORT'*, *log=None*, *report_iter=10*)

Summary Function to perform conjugate gradient descent minimization in Modeller on a user-provided structural file (PDB).

Parameters

struct [str] String for path to PDB file

dest [str] String for path to location where minimized structure will be written

disu [bool] If true, positions of disulfide bridges will be automatically detected

min_atom_shift [float] If the max atomic shift between minimization steps is less than this value, then convergence is reached and minimization is terminated

max_iter [int] Maximum number of calls of objective function before minimization is terminated

output [str] Valid options are 'NO_REPORT' and 'REPORT'. If set to 'REPORT', then a log file during minimation will be printed to screen

log [str or None] String for path to location where minimization report will be saved. If None, no report will be saved. Report contains only values of objective function at after each report interval.

report_iter [int] Integer that describes the number of minimization steps to perform before reporting the objective function.

Returns

mdl [Model object from Modeller] If dest is None, the function will return the minimized model. If dest is specified, then no model will be returned but the minimized model will be written to file.

aesop.aesop.**mutatePDB**(*pdb*, *mutid*, *resnum*, *chain=None*, *resid='ALA'*)

Summary Function to generate a mutant structure given a local PDB file using MODELLER.

Parameters

pdb [str] Full path to pdbfile that will be modified.

mutid [str] Prefix for mutated structure that will be written. May be a full path without file extension if desired output path is not in working directory.

resnum [int, or type that can be forced to int] Integer number specifying residue number to be mutated.

chain [str, optional] Chain ID where specified residue number is to be mutated. This is necessary to specify if residue numbers are not unique on each chain.

resid [str, optional] Three letter amino acid code specifying the type of mutation. Default mutation is to alanine ('ALA').

Returns

None Writes mutated structure to file.

aesop.aesop.**mutatePQR**(*pqrfile*, *mutid*, *resnum*, *chain=None*, *ff='parse'*)

Summary Mutate PQR file via side-chain truncation scheme (mutate to Alanine)

Parameters

pqrfile [str] Full path to PQR file

mutid [str] Prefix to use when writing mutated PQR. Should be a full path if destination is not in working directory.

resnum [int] Residue number to mutate to alanine.

chain [str, optional] Chain where residue that will be mutated is located.

Returns

None Writes mutated PQR to file specified by the prefix mutid.

aesop.aesop.**plotDend**(*esd*, *filename=None*)

Summary Function to display an electrostatic similarity dendrogram from a previously run ElecSimilarity class.

Parameters

esd [ElecSimilarity class] ElecSimilarity class containing final esd matrix.

filename [str, optional] If the resulting plot should be written to disk, specify a filename. Otherwise, the image will only be saved.

Returns

None Writes image to disk, if desired.

aesop.aesop.**plotESD**(*esd*, *filename=None*, *cmap='hot'*)

Summary Function to display an electrostatic similarity heatmap from a previously run ElecSimilarity class.

Parameters

esd [ndarray] ESD matrix from ElecSimilarity class (ElecSimilarity.esd).

filename [str, optional] If the resulting plot should be written to disk, specify a filename. Otherwise, the image will only be saved.

cmap [str, optional] Colormap from matplotlib to use.

Returns

None Writes image to disk, if desired.

aesop.aesop.**plotESD_interactive**(*esd*, *filename=None*, *cmap='YIGnBu'*, *display_output='external'*)

Summary Function to display an electrostatic similarity heatmap from a previously run ElecSimilarity class. Figure is more interactive that the standard matplotlib figure.

Parameters

esd [ElecSimilarity class] ElecSimilarity class containing final esd matrix.

filename [str, optional] If the resulting plot should be written to disk, specify a filename. Otherwise, the image will only be saved.

cmap [str, optional] Colormap from matplotlib to use.

display_output [str] Set output to either open local html file in browser or inline plot in notebook.

Returns

None Writes image to disk, if desired.

aesop.aesop.**plotNetwork**(*scan*, *filename=None*, *title=''*, *dpi=300*, *cutoff=5.0*, *E=2.5*, *node_size=1500*, *font_size=12*, *alpha=0.8*, *edge_color='g'*, *edge_width=3.0*, *layout=None*, ***kwargs*)

Summary Function to visualize electrostatic interactions from a scan class (Alascan or Directed Mutagenesis). Requires networkx to be installed.

Parameters

scan [Alascan or DirectedMutagenesis class] Scan class where calculation of free energies is complete.

filename [str or None] Full path to file where figure will be saved. If None, no figure is saved, but the plot is displayed and the graph is returned.

title [str] Matplotlib style title for plot.

dpi [int] Integer specifying the dots per inch, or image resolution.

cutoff [float] Distance cutoff in Angstroms for determining if a electrostatic interaction occurs. Default value is 5 Angstroms.

E [float] Threshold for determing those nodes that should be included in the network based on the value of the free energy perturbation that results from mutating the amino acid. If the magnitude of the free energy of association relative to the parent structure is greater than E, then the node is included. Default is 2.5 kJ/mol.

node_size [int] Parameter to scale size of nodes in network. Larger values result in nodes with larger diameter.

font_size [int] Font size for text within network. 12 pt font is default.

alpha [float] Set transparency of nodes. Default is 0.8. Accepted range is [0, 1].

edge_color [str] Matplotlib-style specification of line color. Default is 'g' (green).

edge_width [int] Set the line width for edges. Default is 3 pt font.

layout [Networkx layout kernel or None] Network layout from networkx. Extra arguments for this layout may be passed as key word arguments to plotNetwork.

aesop.aesop.**plotNetwork_interactive**(*scan*, *filename=None*, *title=''*, *cutoff=5.0*, *E=2.5*, *font_size=14*, *node_size=20*, *edge_color='#888'*, *edge_width=0.5*, *display_output='external'*, *layout=None*, ***kwargs*)

Summary Function to visualize electrostatic interactions from a scan class (Alascan or Directed Mutagenesis). Figure is more interactive than the standard matplotlib figure. Requires networkx to be installed.

Parameters

scan [Alascan or DirectedMutagenesis class] Scan class where calculation of free energies is complete.

filename [str or None] Full path to file where figure will be saved. If None, no figure is saved, but the plot is displayed and the graph is returned.

title [str] Matplotlib style title for plot.

cutoff [float] Distance cutoff in Angstroms for determining if a electrostatic interaction occurs. Default value is 5 Angstroms.

E [float] Threshold for determing those nodes that should be included in the network based on the value of the free energy perturbation that results from mutating the amino acid. If the magnitude of the free energy of association relative to the parent structure is greater than E, then the node is included. Default is 2.5 kJ/mol.

node_size [int] Parameter to scale size of nodes in network. Larger values result in nodes with larger diameter.

font_size [int] Font size for text within network. 12 pt font is default.

edge_color [str] Matplotlib-style specification of line color. Default is 'g' (green).

edge_width [int] Set the line width for edges. Default is 3 pt font.

display_output [str] Set output to either open local html file in browser or inline plot in notebook.

layout [Networkx layout kernel or None] Network layout from networkx. Extra arguments for this layout may be passed as key word arguments to plotNetwork.

aesop.aesop.**plotScan**(*Alascan*, *filename=None*)

Summary Function to display results from the computational alanine or directed mutagenesis scan.

Parameters

Alascan [scan class] Alascan or DirectedMutagenesis class after running the complete analysis.

filename [None, optional] If the resulting plot should be written to disk, specify a filename. Otherwise, the image will only be saved.

Returns

tuple Handles to generated figure.

aesop.aesop.**plotScan_interactive**(*Alascan*, *display_output='external'*, *filename=None*)

Summary Function to display results from the computational alanine or directed mutagenesis scan. Figure is more interactive than the standard matplotlib figure.

Parameters

Alascan [scan class] Alascan or DirectedMutagenesis class after running the complete analysis.

display_output [str] Set output to either open local html file in browser or inline plot in notebook.

filename [None, optional] If the resulting plot should be written to disk, specify a filename. Otherwise, the image will only be saved.

Returns

None Saves image of figure, if desired.

aesop.aesop.**runProcess**(*command*)

Summary Simple function intended to capture outputs from processes that write to STDOUT.

Parameters

command [list] Lists of strings where each element is a part of the entire command. Ex: ['script','arg1','arg2',...]

Returns

tuple return tuple where first element is output that would have been sent to STDOUT and the second element captures errors.

exception aesop.aesop.**runProcess_Exception**

Bases: exceptions.Exception

aesop.aesop.**superpose**(*ref*, *pdb*, *atype='CA'*, *output=None*)

Summary Uses Modeller to superpose a PDB file (pdb) to a reference PDB (ref).

Parameters

ref [str] Full path to PDB file (or name of file in working directory) that will be used as the reference for superpositioning.

pdb [str] Full path to PDB file (or name of file in working directory) that will be used as the mobile structure for superpositioning.

atype [str] Modeller-compatible string selection for atoms to be used in superpositioning. We suggest using 'CA'.

output [str or None] If output is None, the file specified by pdb will be updated with the superposed structure. If specified, output should be a full path where the superposed structure will be saved.

aesop.aesop.**writePDB**(*alascan*, *filename=None*)

Summary Function to write free energies of association/solvation into B-factor column of PDB for easy visualization of results.

Parameters

alascan [scan class] Alascan or DirectedMutagenesis class after running the complete analysis.

filename [str, optional] Full path to file where PDB file will be written. Defaults to job directory.

Bibliography

[Harrison2016] Harrison REH, Mohan RR, Gorham RD Jr, Kieslich CA, Morikis D (2016, in submission) AESOP: A Python Library for Investigating Electrostatics in Protein Interactions

[Kieslich2011-1] Kieslich, C.A., R.D. Gorham, and D. Morikis. 2011. Is the rigid-body assumption reasonable?: Insights into the effects of dynamics on the electrostatic analysis of barnase-barstar. J. Non. Cryst. Solids. 357: 707?716..

[Gorham2011-1] Gorham, R.D., C.A. Kieslich, and D. Morikis. 2011. Electrostatic clustering and free energy calculations provide a foundation for protein design and optimization. Ann. Biomed. Eng. 39: 1252?1263..

[Gorham2011-2] Gorham, R.D., C.A. Kieslich, A. Nichols, N.U. Sausman, M. Foronda, and D. Morikis. 2011. An evaluation of Poisson-Boltzmann electrostatic free energy calculations through comparison with experimental mutagenesis data. Biopolymers. 95: 746?754..

[Kieslich2011-2] Kieslich, C.A., D. Morikis, J. Yang, and D. Gunopulos. 2011. Automated computational framework for the analysis of electrostatic similarities of proteins. Biotechnol. Prog. 27: 316?325..

[Chen2015] Chen C, Gorham RD Jr., Gaieb Z, and Morikis D (2015) Electrostatic interactions between complement regulator CD46(SCR1-2) and adenovirus Ad11/Ad21 fiber protein knob, Molecular Biology International, 2015: Article ID 967465. 15 pages. DOI:10.1155/2015/967465.

[Harrison2015] Harrison RES, Gorham RD Jr, Morikis D (2015) Energetic evaluation of binding modes in the C3d and Factor H (CCP 19-20) complex, Protein Science 24:789-802. DOI:10.1002/pro.2650.

[Mohan2015] Mohan R, Gorham RD Jr, Morikis D (2015) A theoretical view of the C3d:CR2 binding controversy, Molecular Immunology 64:112:122. DOI:10.1016/j.molimm.2014.11.006.

[Liu2014] Liu Y, Kieslich CA, Morikis D, Liao J (2014) Engineering pre-SUMO4 as efficient substrate of SENP2, Protein Engineering Design & Selection 27:117-126. DOI: 10.1093/protein/gzu004.

[Gorham2014] Gorham RD Jr, Rodriguez W, Morikis D (2014) Molecular analysis of the interaction between staphylococcal virulence factor Sbi-IV and complement C3d, Biophysical Journal 106:1164-1173. DOI: 10.1016/j.bpj.2014.01.033.

[Kieslich2012] Kieslich CA, Morikis D (2012) The two sides of complement C3d: evolution of electrostatics in a link between innate and adaptive immunity, PLoS Computational Biology 8:e1002840 (8 pages). DOI: 10.1371/journal.pcbi.1002840.

[Bellows-Peterson2012] Bellows-Peterson ML, Fung H, Floudas CA, Kieslich CA, Zhang L, Morikis D, Wareham KJ, Monk PN, Hawksworth O, Woodruff TM (2012) De novo peptide design with C3a receptor agonist and antagonist activities: theoretical predictions and experimental validation, Journal of Medicinal Chemistry 55:4159-4168.

[Gorham2012] Gorham Jr RD, Kieslich CA, Morikis D (2012) Complement inhibition by Staphylococcus aureus: electrostatics of C3d-EfbC and C3d-Ehp association, Cellular and Molecular Bioengineering 5:32-43.

[El-Assaad2011] El-Assaad AM, Kieslich CA, Gorham Jr RD, Morikis D (2011) Electrostatic exploration of the C3d-FH4 interaction using a computational alanine scan, Molecular Immunology 48:1844-1850. Erratum (2013) 53:173-174.

[Hakkoymaz2011] Hakkoymaz H, Kieslich CA, Gorham Jr RD, Gunopulos D, Morikis D (2011) Electrostatic similarity determination using multi-resolution analysis, Molecular Informatics 30:733-746.

[Kieslich2011-3] Kieslich CA, Vazquez H, Goodman GN, L?pez de Victoria A, Morikis D (2011) The effect of electrostatics on Factor H function and related pathologies, Journal of Molecular Graphics and Modeling 29:1047-1055.

[Chae2010] Chae K, Gonong BJ, Kim SC, Kieslich CA, Morikis D, Balasubramanian S, Lord EM (2010) A multifaceted study of stigma/style cysteine-rich adhesion (SCA)-like Arabidopsis lipid transfer proteins (LTPs) suggests diversified roles for these LTPs in plant growth and reproduction, Journal of Experimental Botany 61:4277-4290.

[Chae2010] Chae, K., B.J. Gonong, S.C. Kim, C.A. Kieslich, D. Morikis, S. Balasubramanian, and E.M. Lord. 2010. A multifaceted study of stigma/style cysteine-rich adhesin (SCA)-like Arabidopsis lipid transfer proteins (LTPs) suggests diversified roles for these LTPs in plant growth and reproduction. J. Exp. Bot. 61: 4277–4290.

[Kieslich2011-2] Kieslich, C.A., R.D. Gorham, and D. Morikis. 2011. Is the rigid-body assumption reasonable?: Insights into the effects of dynamics on the electrostatic analysis of barnase-barstar. J. Non. Cryst. Solids. 357: 707–716..

[Gorham2011-2] Gorham, R.D., C.A. Kieslich, A. Nichols, N.U. Sausman, M. Foronda, and D. Morikis. 2011. An evaluation of Poisson-Boltzmann electrostatic free energy calculations through comparison with experimental mutagenesis data. Biopolymers. 95: 746–754..

[Kieslich2011-2] Kieslich, C.A., R.D. Gorham, and D. Morikis. 2011. Is the rigid-body assumption reasonable?: Insights into the effects of dynamics on the electrostatic analysis of barnase-barstar. J. Non. Cryst. Solids. 357: 707–716..

[Gorham2011-2] Gorham, R.D., C.A. Kieslich, A. Nichols, N.U. Sausman, M. Foronda, and D. Morikis. 2011. An evaluation of Poisson-Boltzmann electrostatic free energy calculations through comparison with experimental mutagenesis data. Biopolymers. 95: 746–754..

Python Module Index

a

aesop.aesop, 21

Index

A

aesop.aesop (module), 21
Alascan (class in aesop.aesop), 21
APBS_Exception, 21

B

batchAPBS() (in module aesop.aesop), 36
batchCalcDX() (in module aesop.aesop), 36
batchCoulomb() (in module aesop.aesop), 36

C

calcAPBS() (aesop.aesop.Alascan method), 24
calcAPBS() (aesop.aesop.DirectedMutagenesis method), 29
calcAPBS_parallel() (aesop.aesop.Alascan method), 24
calcAPBS_parallel() (aesop.aesop.DirectedMutagenesis method), 29
calcCoulomb() (aesop.aesop.Alascan method), 24
calcCoulomb() (aesop.aesop.DirectedMutagenesis method), 30
calcCoulomb_parallel() (aesop.aesop.Alascan method), 24
calcCoulomb_parallel() (aesop.aesop.DirectedMutagenesis method), 30
calcDX() (in module aesop.aesop), 37
calcESD() (aesop.aesop.ElecSimilarity method), 34
calcESI() (aesop.aesop.Alascan method), 24
calcESI() (aesop.aesop.ElecSimilarity method), 34
centerPDB() (aesop.aesop.ElecSimilarity method), 34
checkerrors() (aesop.aesop.Alascan method), 25
checkerrors() (aesop.aesop.DirectedMutagenesis method), 30
checkerrors() (aesop.aesop.ElecSimilarity method), 34
checkwarnings() (aesop.aesop.Alascan method), 25
checkwarnings() (aesop.aesop.DirectedMutagenesis method), 30
checkwarnings() (aesop.aesop.ElecSimilarity method), 34
Complete_PDB_Exception, 27
complete_structure() (in module aesop.aesop), 37

D

ddGa_rel() (aesop.aesop.Alascan method), 25
ddGa_rel() (aesop.aesop.DirectedMutagenesis method), 30
dGsolv_rel() (aesop.aesop.Alascan method), 25
dGsolv_rel() (aesop.aesop.DirectedMutagenesis method), 30
DirectedMutagenesis (class in aesop.aesop), 27

E

ElecSimilarity (class in aesop.aesop), 32
execAPBS() (in module aesop.aesop), 37
execCoulomb() (in module aesop.aesop), 38
execPDB2PQR() (in module aesop.aesop), 38

F

find_grid() (aesop.aesop.Alascan method), 25
find_grid() (aesop.aesop.DirectedMutagenesis method), 30

G

genDirs() (aesop.aesop.Alascan method), 25
genDirs() (aesop.aesop.DirectedMutagenesis method), 30
genDX() (aesop.aesop.ElecSimilarity method), 34
genDX_parallel() (aesop.aesop.ElecSimilarity method), 34
genMutid() (aesop.aesop.Alascan method), 25
genMutid() (aesop.aesop.DirectedMutagenesis method), 30
genParent() (aesop.aesop.Alascan method), 25
genParent() (aesop.aesop.DirectedMutagenesis method), 31
genPDB() (aesop.aesop.DirectedMutagenesis method), 31
genPQR() (aesop.aesop.DirectedMutagenesis method), 31
genPQR() (aesop.aesop.ElecSimilarity method), 35

genTruncatedPQR() (aesop.aesop.Alascan method), 26
getMutids() (aesop.aesop.Alascan method), 26
getMutids() (aesop.aesop.DirectedMutagenesis method), 31
Grid (class in aesop.aesop), 35

I

initializeGrid() (aesop.aesop.ElecSimilarity method), 35

M

minimize_cg() (in module aesop.aesop), 38
Minimize_CG_Exception, 36
minimPDB() (aesop.aesop.ElecSimilarity method), 35
mutatePDB() (in module aesop.aesop), 39
mutatePQR() (aesop.aesop.ElecSimilarity method), 35
mutatePQR() (in module aesop.aesop), 39

P

PDB2PQR_Exception, 36
plotDend() (in module aesop.aesop), 40
plotESD() (in module aesop.aesop), 40
plotESD_interactive() (in module aesop.aesop), 40
plotNetwork() (in module aesop.aesop), 40
plotNetwork_interactive() (in module aesop.aesop), 41
plotScan() (in module aesop.aesop), 42
plotScan_interactive() (in module aesop.aesop), 42

R

re (aesop.aesop.Grid attribute), 36
readDX() (aesop.aesop.Grid method), 36
run() (aesop.aesop.Alascan method), 26
run() (aesop.aesop.DirectedMutagenesis method), 31
run() (aesop.aesop.ElecSimilarity method), 35
run_parallel() (aesop.aesop.Alascan method), 26
run_parallel() (aesop.aesop.DirectedMutagenesis method), 31
run_parallel() (aesop.aesop.ElecSimilarity method), 35
runProcess() (in module aesop.aesop), 42
runProcess_Exception, 42

S

set_grid() (aesop.aesop.Alascan method), 26
set_grid() (aesop.aesop.DirectedMutagenesis method), 31
summary() (aesop.aesop.Alascan method), 26
summary() (aesop.aesop.DirectedMutagenesis method), 32
superpose() (in module aesop.aesop), 42
superposePDB() (aesop.aesop.ElecSimilarity method), 35

V

viewLogs() (aesop.aesop.Alascan method), 27
viewLogs() (aesop.aesop.DirectedMutagenesis method), 32
viewLogs() (aesop.aesop.ElecSimilarity method), 35

W

writeDX() (aesop.aesop.Grid method), 36
writeLogs() (aesop.aesop.Alascan method), 27
writeLogs() (aesop.aesop.DirectedMutagenesis method), 32
writeLogs() (aesop.aesop.ElecSimilarity method), 35
writePDB() (in module aesop.aesop), 43