Dockeye Howto. Kim Sharp, Univ. of Pennsylvania, Sept. 2019

Installation

1) download dockeye_multi from github, and rename the top directory: dockeye_multi

lets say your dockeye_multi full path name is: /Users/<username>/source/dockeye_multi/

where <username> is your username.

2) change directory (cd) to /Users/<username>/source/dockeye_multi/src and create a soft link to the correct shared object file there. e.g. on MacOS:

ln -s -i dockeyeM_energy_MacOS_python27_64bit.so dockeyeM_energy.so

3) tell python to look in this directory for import modules:

bash: export PYTHONPATH="/Users/<username>/source/dockeye_multi/src/"

csh: setenv PYTHONPATH /Users/<username>/source/dockeye_multi/src/

Going forward you should put this in your login shell script. If you are using the bash shell: in .bashrc or .profile or .bashrc_profile (??). If you are using the csh shell: in .cshrc. PYTHONPATH will be set automatically when you login/open a terminal window. To check, start up an interactive python2.7 session by typing on the command line:

python2.7

and type:

import dockeye_methods

If there are no error messages, your environment is set, so quit python.

File prepping example

To run dockeye_multi, at the very least you need to assign a radius to each atom. This will allow you to dock based solely on 'shape', a limited but useful filter for generating possible poses. More realistic docking includes electrostatic effects, which requires assigning a charge to each atom. The radius and the charge (even if equal to 0.0) must be placed in the occupancy and b-factor fields, respectively, of a pdb format file. In addition, to dock a flexible

ligand, all the required conformations must be pre-generated and placed in the ligand pdb file, each delimited by MODEL/ENDMDL records. If there is just one conformer, it should still be delimited by MODEL/ENDMDL records.

Atomic radii typically vary little between forcefields. A 'standard set' are assigned on the fly by dockeye_multi. They are contained in a dictionary atom_rads{} in dockeye_methods.py, so if you don't like them feel free to edit the dictionary. There are many ways to assign atomic charges, and many charge parameter sets (forcefields). Every molecular dynamics package has tools and parameter files to do this. The electrostatics software packages DelPhi and APBS also have comparable tools. The protein target is usually no problem. There are quite a few well tested protein charge sets. The difficulty is usually the ligand, which often contains novel chemical groups and bonded configurations of atoms. No claim is made about the charge assignment method here using autodock tools, (http://autodock.scripps.edu/) except it is robust, easy (compared to QM!), and it will allow you to get to the actual dockeye_multi part quickly to learn how to use it.

Importantly, the autodock tool adt adds hydrogen atoms (another annoying pre-step required in many modelling efforts!), and it does a nice job of identifying rotatable torsion bonds. This is an essential step in pre-generating the ligand conformers.

Workflow Example

The following tutorial example work flow, using adt and several helper programs that come with dockeye_multi, will take you from an initial pdb file (e.g. downloaded from the Protein Data Bank) through a dockeye_multi/PyMol session.

1) Download the target protein structure in pdb format from www.rcsb.org. Cut out a single biological protein unit (minus water, ions, crystallization compounds etc) into a separate file. If there is a co-crystallized ligand present, cut it out into a second file OR obtain the ligand structure from other sources (csd database, smiles string etc) and convert to pdb format. (Obviously in the first case, we know the true pose, so this would correspond to a 'training' session, in which we see how close to the correct pose we can get. In the second case, we remove any xtal structure ligand, if present, and try to dock a different ligand, so this is the 'test' or 'real' case)

2) Run dockeye_prep.py on both files to position them nicely in the same coordinate frame:

python $PYTHONPATH/dockeye_prep.py <protein_filename> <ligand_filename>

output will be two files:

de_prot.pdb, de_ligand.pdb

3) run autodock tools (adt) on protein file de_prot.pdb to add hydrogens and atomic charges:

file→ read molecule→ de_prot.pdb

edit → hydrogens → add → polar only

edit → atoms → assign ad4 type

edit → charges → add kollman charges

file → save → write pdbqt

output file is de_prot.pdbqt

4) put radii and charges into occupancy and b-factor files of protein pdb file:

python $PYTHONPATH/pdbqt_to_pdb.py de_prot.pdbqt

output file de_prot_qr.pdb is ready for use in dockeye_multi

5) run autodock tools (adt) on the ligand file to add hydrogens and atomic charges, and then use the ligand option identify rotatable bonds:

file→ read molecule→ de_ligand.pdb

edit → hydrogens → add → polar only

edit → atoms → assign ad4 type

edit → charges → add kollman charges

ligand → input → choose → de_ligand → select molecule for autodock4

ligand → output → save as pdbqt

output file is de_ligand.pdbqt

6) put radii and charges into occupancy and b-factor files of ligand pdb file:

python $PYTHONPATH/pdbqt_to_pdb.py de_ligand.pdbqt

output file de_ligand_qr.pdb

7a) run make_conformers.py on the ligand file to pre-generate ligand conformers

python $PYTHONPATH/make_conformers.py de_ligand_qr.pdb

output file de_ligand_qr_tor.pdb is ready for use in dockeye

7b) (Alternate). Run make_conformers_flip.py on the ligand file to pre-generate ligand conformers.

python $PYTHONPATH/make_conformers_flip.py de_ligand_qr.pdb

output file de_ligand_qr_torf.pdb is ready for use in dockeye

make_conformers_flip.py identifies the principle moments of inertia of the ligand, and in addition to the conformers generated in 7a, generates three copies of each, flipped 180 degrees around each principle MOI. This can be useful for rather flat ligands which are approximately symmetric, as dockeye will automatically test each of the 4 flipomers for each pose and display the best one, saving the user time, but requring 4 times as much computation

8) using text editor create pymol script file, setup.pml

with pymol/dockeye commands in it:

#---

run $PYTHONPATH/dockeyeM_c.py

de("de_prot_qr.pdb","de_ligand_qr_tor.pdb")

#optional view settings

hide lines

spectrum b, red_white_blue

show sticks, dockeye_lig

show surface, de_prot_qr

set transparency, 0.4

#---

Running dockeye_multi

at the command line type:

pymol setup.pml

inside pymol set: Mouse mode → 3-button editing

mouse over the ligand, hold the shift key down to rotate/translate ligand relative to the protein, using mouse motions. Release the shift key to do global rotate/translate. You really need a 3-button mouse for this, trackpads do not work well.

The dockeye_multi plugin

The dockeye plugin creates new graphical elements that react as you move the ligand with respect to the protein:

A

Three protein-ligand interaction energy bars on the left: electrostatic, van der Waals, and total energies. (red = positive energy, blue = negative energy)

There is a green 'low water mark' which marks the best (most -ve) total energy so far.. Each

time a new low is discovered, the pose is written to a date/time-stamped log file.

(ligand pdb files can be generated later from these stored poses by running dockeyeM_getpose.py on the log file and selecting any of the poses)

B

the 'dockeye object' named in the pymol object menu **dockeye_obj**:

This is a set of colored circles characterizing the interaction between ligand and protein. For every ligand-protein atomic pair interaction above some absolute energy cutoff (+-ve or -ve), a circle is generated. Its position is midway between the atom pairs, oriented perpendicular to line joining the two atoms. Its radius varies inversely with distance, and it is color coded on a red-green-blue scale according to the sign and magnitude of the pair interaction total energy (red: +ve, blue -ve)

C

The ligand pose and conformer is shown in stick representation, overlaid on the original conformer. For each pose dockeye_multi's back end evaluates the ligand-protein interaction energy for all the pre-generated conformers and displays the one with the best interaction (lowest energy)

In brief, the 'Game' you are playing is to move that green bar downwards, by surrounding the ligand by as many blue circles as you can get, and avoiding those red circles! The number, location and color of the dockeye circles give you visual cues as to where the ligand is making good/bad interactions, or, as importantly, is lacking interactions, and how to move the ligand to improve the pose.

Extracting a ligand pose

For examples, if a dockeye run created the following logfile:

dockeyeM_17Sep2019_16:29:25.log.sav

type the following on the command line:

python $PYTHONPATH/dockeyeM_getposes.py dockeyeM_17Sep2019_16:29:25.log.sav

The program will write out all poses as a 'movie' pdb file, <pdbfilename>_allposes.pdb, and

pause. You can read the movie file into PyMol and click through using the playback buttons to help select the pose. Poses are also listed by energy. Select your pose by number and exit. Example output file: **de_ligand_qr_tor_pose_5.pdb**

The tryp directory in the example folder demonstrates this work flow for 3ptb.pdb from www.rcsb.org. Here we re-dock the benzamidine inhibitor back into trypsin. Use the file time stamps to follow the workflow.