dockeye Howto.

Kim Sharp, Univ. of Pennsylvania, Feb 2021

Table of Contents

Installation

1) Download dockeye_multi from github and rename the top directory: **dockeye_multi** if necessary

2) Change directory to dockeye_multi/src and create soft links to the correct back end shared object file and front end Python code for your operating system and PyMol/Python version.

```
ln -sf <dir>/dockeyeM_energy<build_version>.so dockeyeM_energy.so
```

```
ln -sf <dir>/dockeyeM_<version>.py dockeyeM_c.py
```

Text within <> in the above commands varies depending on platform/PyMol version: See details in the INSTALL file.

How to cite dockeye in published work

Baskaran, S. G.; Sharp, T. P.; Sharp, K. A. Computational Graphics Software for Interactive Docking and Visualization of Ligand–Protein Complementarity. J. Chem. Inf. Model. 2021. https://doi.org/10.1021/acs.jcim.0c01485.

File Prepping

To run dockeye_multi, one input pdb-format file is needed for the target (usually protein), and one input pdb-format file is needed for the ligand. A radius and a charge needs to be assigned to each atom in both of these files. Depending on the charge set to be used and the source of the files, coordinates and pdb records for hydrogen atoms may have to generated first.

Atomic radii

Atomic radii typically vary little between forcefields. A 'standard set' derived from Bondi, A. (1968) "Molecular Crystals, Liquids and Glasses" are assigned on the fly by dockeye_multi when it reads in the input pdb files. The radii are contained in a dictionary **atom_rads{}** in **dockeye_methods.py**, so if you don't like them feel free to edit the dictionary.

Atomic charges

The atomic charge for each atom (even if equal to 0.0) must be placed in the B-factor field (columns 62-66) of the pdb format input files of both target and ligand file. Setting all the atomic charges to zero in either the target or ligand, or switching off the electrostatics (using the charges=False option at dockeye startup – see below) 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. This must be done prior to docking by placing the charge values into the protein and ligand input files.

Flexible ligand docking

To dock a flexible ligand all the required conformations must be pre-generated and placed in the input ligand pdb file, each conformer delimited by MODEL/ENDMDL records. Atomic charges need to be present for each atom of each conformer (thus they need not be the same for each conformer, in this way allowing for conformation-dependent polarization effects.) If there is just one conformer, it must still be delimited by MODEL/ENDMDL records.

Helper programs

Strictly speaking, charge assignment and conformer generation is not part of the dockeye_multi package *per se*, although various helper programs are supplied. 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 targets are 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. If you have *.psf ('protein structure format') files generated by MD programs such as CHARMM or NAMD, you can extract these charges into your target and ligand pdb files using the helper Python program **psfpdb_to_atm.py** provided in the src directory. The examples provided with dockeye_multi, illustrated below, used autodock tools (**adt**) (http://autodock.scripps.edu/) No claim is made here about docking accuracy using this charge assignment method, except that for a wide variety of ligands it is robust, fast & 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 if your pdb files don't have them (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 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). In what follows we assume that the environment variable DOCKEYE points to the src sub-directory in the main dockeye directory.

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

```
python $DOCKEYE/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 gasteiger 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 $DOCKEYE/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 gasteiger 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 $DOCKEYE/pdbqt_to_pdb.py de_ligand.pdbqt
```

output file is de_ligand_qr.pdb

7a) Depending on the number of rotatable bonds, (given e.g. by the TORSDOF entry at the end of **adt**'s output pdbqt file) run either **make_conformers.py** (TORSDOF <= 5) or **random_conformers.py** on the ligand file to pre-generate ligand conformers. **make_conformers.py** will generate all $3^{(TORSDOF)}$ gauche-, gauche+, trans conformers, which hits the practical limit for dockeye_multi using typical workstation cpus of 243 conformers for 5 rotatable bonds. **random_conformers.py** will generate the requested number of conformers by randomly sampling gauche-, gauche+, trans conformers, doing a simple bump check, and discarded bad conformers for better ones. No claim to an in-depth production of high quality, low energy conformers representative of highly populated species is claimed by these methods.

```
python $DOCKEYE/make_conformers.py  de_ligand_qr.pdb
```

output file **de_ligand_qr_tor.pdb** is ready for use in dockeye

```
python $DOCKEYE/random_conformers.py  de_ligand_qr.pdb  100
{True/False}
```

output file **de_ligand_qr_ran.pdb** contains 100 conformers and is ready for use in dockeye. If the optional True/False flag is set to True, then for each conformer generated by random_conformers.py, three 'flipmers' will be generated, each flipmer being rotated 180^{o} around one of the three principle ellipsoid of inertia axes. At the cost of 4-fold more conformers, the three flipmers aid in the exploration of pose space, which can be useful for ligands with any appreciable two-fold symmetry axes.

7b) (Alternate). Run **make_conformers_flip.py** on the ligand file de_ligand_qr_tor.pdb to pre-generate exhaustive ligand conformers plus their three flipmers.

```
python $DOCKEYE/make_conformers_flip.py  de_ligand_qr.pdb
```

8) Using a text editor create a pymol script file named setup.pml

with the following PyMol/dockeye commands in it (or copy setup.pml from one of the example directories and edit to suit)

START of setup.pml----------

run <full_dir_pathname>/dockeye_multi/src/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, dockeye_prt

set transparency, 0.4

END of setup.pml ----------

Only the run and de() lines are required- the rest are optional,but produces a convenient display for docking. Note that you will have to replace the text inside <> with the full directory pathname where you placed the dockeye_multi files.

Running dockeye_multi

at the command line type:

```
pymol setup.pml
```

to start up PyMool, load the protein, ligand and the dockeye plugin

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 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 line representation, along with the original conformer rendered in stick for easy 'grabbing' with the mouse. 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)

Docking in real time

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, orientation 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.

Bookmarking poses

Dockeye will automatically log any pose that has a new energy minimum (for the current pymol session) to the log file for subsequent retrieval. In addition, the user may bookmark any pose, regardless of its energy. In the pymol text box entry type:

mark(#)

where # is any single digit number from 1-9. Say you type:

mark(2)

Dockeye will then write a ligand file out in current pose named:

dockeye_lig_mark_2.pdb

and a pymol setup file called:

dockeye_mark_2.pml

you can then start a new pymol session (the first one can still be running) by typing:

pymol dockeye_mark_2.pml

at the command line in a new terminal window and you startup with the bookmarked pose. You can bookmark up to 9 different poses. If you want more bookmarks, just rename the files so they don't get clobbered.

Extracting a ligand pose from the logfile

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

`dockeyeM_17Sep2019_16:29:25.log`

type the following on the command line:

```
python $DOCKEYE/dockeyeM_getposes.py dockeyeM_17Sep2019_16:29:25.log
```

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**

Command line options for dockeye_multi

After loading the dockeye module dockeyeM_c.py with the run command (either from a PyMol script file at startup or at the PyMol command line) dockeye_multi is invoked inside PyMol by the **de()** command (again either from a PyMol script file at startup or at the PyMol command line. The **de()** command takes two mandatory string arguments, namely the names of the input target and ligand files, "de_prot_qr.pdb" and "de_ligand_qr_tor.pdb" in the example above. Additional optional arguments may be given at invocation time. These, with default values are

charges=True (charges=False will set all charges to 0.0 so only the vdw energy is used)

logscale=True (energy bars are shown using a logscale – this keeps them within the PyMol window better, but it is harder to read 'absolute' values off the bars – however numeric values are written to console and log file)

dielectric=80. (bulk dielectric used for electrostatics – any floating point value > 0. is valid)

eps=0.1 (depth of van der waals potential in kcal/mole. any floating point value >= 0. is valid. Note the a 9-6 form is used for the vdd potential. Softening the repulsive term from the usual $1/r^{12}$ facilitates the finding of poses in tight pockets, and compensates to a small degree for the rigid protein approximation)

handle=False (the original conformer will always be displayed in stick representation, while the current best conformer is diplayed in line represention. Displaying a persistent stick object aids in 'grabbing' the ligand with a mouse/shift-key in edit mode, so as to move it – In Pymol I have found this is simply easier to do with a stick vs. line object. Setting handle=True will replace the stick-conformer by a three-dimensional cross in stick representation. Depending on the moves, views and ligand structure, sometimes it is easier to manipulate the ligand

pose with one form of 'grab-handle' than another.

Fully worked example

The tryp directory in the example folder contains files from this complete work flow for 3ptb.pdb from www.rcsb.org. Here we re-docked the benzamidine inhibitor back into trypsin, and created one bookmark along the way. Use the file time stamps to follow the workflow.