Rosetta ligand docking tutorial

true true

original: 2017 - (last updated by JYS: 2024-09-03)

Abstract

This is a "rewrite" of the Rosetta `ligand_docking_tutorial.md` with comments on how to combine Docker Container and macOS.

Contents

1 Rewrite Summary

This is a "rewrite" of the Rosetta `ligand_docking_tutorial.md` file in R-markdown with comments on how to combine Docker Container and macOS. (There are no Windows bina-

ries.)

Summary: Combine software and scripts on Docker and local macOS computer (Intel amd64 or arm64 Silicon Chip M series) to follow successfully the Rosetta tutorial Ligand Docking with a G-Protein Coupled Receptor. This method will allow access to the native OS speed while fulfilling all preparatory and exploratory steps that fail or are too complex to set-up on the local computer.

2 Rosetta

> The Rosetta software suite includes algorithms for computational modeling and analysis of protein structures. [...] including de novo protein design, enzyme design, ligand docking, and structure prediction of biological macromolecules and macromolecular complexes.

The software is rather complex and it can be difficult to "make things work" considering the breadth of options for algorithms or hardware and operating system support. This method will allow to access the native OS speed while fulfilling all preparatory and exploratory steps.

This post is an attempt to help users that want to use the software on their native OS (macOS) but since some functionality built-in the tutorials assumes a Linux OS, some of the steps are in fact easier handled on the Linux side thanks to Docker.

3 Preparations

These instructions should benefit macOS users primarily as there are no Windows binaries available. Windows users should use the WSL2 method to install Linux under Windows.

Users should be somewhat familiar with bash command line (see *e.g.* my Survival Command Line tutorial) and have the Docker Desktop software installed (see *e.g.* my tutorial Docker – Beginner for Biologists.)

The Rosetta Docker image can be downloaded from **`Terminal`** with the command below, assuming that Docker is already installed:

```
docker pull rosettacommons/rosetta:latest
```

While the purpose of the Docker image is to provide access to all the Linux compiled binaries, we will take advantage of some of the Linux functionality as well as the installed Python within.

3.1 Rosetta preparations

Rosetta is freely available for academic and non-commercial purposes, under license. The software can be downloaded from the links provided on the Download page.

In order to compute the docking computation "natively" (for faster results) on the local computer users should download the newest "release", *e.g.* from the Academic download page.

For this post I used Rosetta 3.14 for M1 (Silicon Chip "M1 binaries", 13Gb) Macintosh, Intel-based Mac users should download the "Mac binaries" (14Gb).

> ***Note***: Unarchiving the file will require about 45 Gb of disk space but will contain the material for all tutorials and demos.

3.2 Getting Started

We will use 2 Terminal sessions: one to navigate within the Macintosh natively. The other to run a Docker container that will be activated in a way so that both Terminal sessions will share the same directory area on the local computer.

When `Terminal` is running you can choose a different color for each shell using the **Shell > New Window** menu cascade. (Basic is white background.)

Here we'll use a blue background for when we are looking ***within*** the Docker Container, and a pale yellow when we are on the macOS side:

```
echo pale yellow means macOS side
```

A pale yellow Terminal can be obtained with the menu cascade:
Shell > New Window > Man Page

```
echo light blue means we are running from inside the Linux container
```

A blue background (with white text) can be obtained with the menu cascade:
Shell > New Window > Ocean

> *Note*: the colored background will only be visible in the HTML versions. (PDF versions are better for printing.)

3.2.1 Environment variables on macOS Terminal

On the Macintosh side it will be useful to create environment variables as suggested in the Rosetta Commons section How To Read These Tutorials. Assuming that the binaries are found within the `Downloads` directory, we can keep that location and its default name.

For the *M1 series* the unarchived directory was called `rosetta.binary.m1.release-371` and the following variables below were created. I replaced my username by `$USER` so that these commands become generic and can be copied, with the caveat that the binary name might be different (change accordingly!)

Open a Mac Terminal (`/Applications/Utilities/Terminal.app`) and paste the (edited) commands:

```
export ROSETTA3=/Users/$USER/Downloads/rosetta.binary.m1.release-371/main/source
export ROSETTA3_DB=/Users/$USER/Downloads/rosetta.binary.m1.release-371/main/database
export ROSETTA_TOOLS=/Users/$USER/Downloads/rosetta.binary.m1.release-371/main/tools
export ROSETTA3_DEMOS=/Users/$USER/Downloads/rosetta.binary.m1.release-371/main/demos
```

These commands can me made "permanent" by including them within the `.zshrc` or `.bashrc` files for the `zsh` or `bash` shell choice respectively. (You can know your shell with command `echo $SHELL`)

3.2.2 Start the Docker container

Open a new Terminal and select a different color to better distinguish this Terminal with the Top menu cascade: **Shell > New Window > Ocean**

Navigate the the top level of the release directory within the main directory. For me it would be as follows...

```
cd /Users/$USER/Downloads/rosetta.binary.m1.release-371/main/
```

Note: the container is not yet running, hence this command is printed with a pale yellow background *i.e.* we are still on the macOS side.

Verify that this was successful with `pwd and` then continue.

Launch the Docker container: The `-v` option lets us share the current directory (*i.e.* `main`, with the container, mapping it as `/data` within the container, and making it the default *working directory* with the `-w` option.

This command is given to macOS. Once we are running, we'll be in the "blue" session *i.e.* ***within*** the Linux OS.

```
docker run -it --rm -v ${PWD}:/data -w /data rosettacommons/rosetta
```

(Note: on a Silicon M series Mac Docker will complain about the "platform" but this can be ignored.)

A listing of the files and directories should reveal the same content as the main directory. We are now ***within*** the Linux container and therefore we'll use blue background.

The shell prompt will be shown as `#` preceded by the working directory as `/data #` which is omitted for easier copy/paste of the commands.

```
ls -F
```

The result should be similar to the following:

```
CITING_ROSETTA.md     README.md           rosetta_scripts_scripts/
CLA.md                database/           source/
CONTRIBUTING.md       demos/              tests/
LICENSE.md            documentation/      tools/
PyRosetta.notebooks/  pyrosetta_scripts/
```

We can check a couple more things: the shell as well as the operating system:

```
echo $SHELL
```

```
/bin/bash
```

```
cat /etc/os-release
```

Will show that we are running Ubuntu:

```
NAME="Ubuntu"
VERSION="20.04.6 LTS (Focal Fossa)"
ID=ubuntu
ID_LIKE=debian
PRETTY_NAME="Ubuntu 20.04.6 LTS"
VERSION_ID="20.04"
HOME_URL="https://www.ubuntu.com/"
SUPPORT_URL="https://help.ubuntu.com/"
BUG_REPORT_URL="https://bugs.launchpad.net/ubuntu/"
PRIVACY_POLICY_URL="https://www.ubuntu.com/legal/terms-and-policies/privacy-policy"
VERSION_CODENAME=focal
UBUNTU_CODENAME=focal
```

Within the Docker container we don't really need the `ROSETTA` environment variables as all paths can start with `/data` to be "absolute" (*i.e.* non ambiguous.)

3.2.3 Docker Container additional set-up

The tutorial assumes that the user is sitting in front of the fully functional Linux computer, including graphical interface, with all ancillary software needed for such a computer already installed. However, within the container a few important utilities are missing, so we need to add them now.

The command `whoami` will confirm that we are running as `root` as is indicated by the `#` prompt. This means that we can install any software we need. The first command provides Ubuntu with information needed to know **where** to download the software we ask to install. On the second line we ask to install (yes by default with `-y`) 3 software

Issue the following commands after the # prompt:

```
apt-get update
apt-get install -y wget nano pymol
```

With these installed we can run all of the commands in the tutorial. Adding the `pymol` Python module will allow the creation of PyMOL `.pse` files as well as a specific python script, as described in the tutorial, from the Docker Container Terminal.

4 Following the original tutorial

This document is a modification of the original tutorial Ligand Docking with a G-Protein Coupled Receptor with added information. A short version is available in the Blog entry about this specific tutorial. This document contains all original writing.

From this point below we'll start using the original tutorial with the following modifications:

1. Both macOS and Linux (Container) commands will be shown if it is possible to run on either side.

2. On macOS the `$ROSETTA` environment variables will be used. On Linux we can simply use `/data`. The replace `<path-to-Rosetta>` in the original commands.
3. We'll continue coloring the command bacground to help indicate the OS used.

4. The ***section numbering*** will mimic as closely as possible the numbering of the original document.

5 Modified Tutorial

From this point the text is mostly from the original tutorial, with added options to run from macOS or Linux as well as additional explanations on the commands used.

KEYWORDS: LIGANDS DOCKING

Bold text means that these files and/or this information is provided.

Italicized text means that this material will NOT be conducted during the workshop

`fixed width text means you should type the command into your terminal (after > sign)`

If you want to try making files that already exist (*e.g.*, input files), write them to a different directory! (`mkdir my_dir`)

In addition to following this sample docking problem, the user is encouraged to review the Rosetta user guide including the section on ligand-centric movers for use with RosettaScripts.

https://www.rosettacommons.org/docs/latest/

5.1 Ligand Docking with a G-Protein Coupled Receptor

The experimental data for this tutorial is derived from: **Chien, E. Y. T. et al. Structure of the human dopamine D3 receptor in complex with a D2/D3 selective antagonist. Science 330, 1091-5 (2010).**

This particular D3/eticlopride protein-ligand complex was used as a target in the GPCR Dock 2010 assessment, the results of which are discussed here: **Kufareva, I. et al. Status of GPCR modeling and docking as reflected by community-wide GPCR Dock 2010 assessment. Structure 19, 1108-1126 (2011).**

If you are interested in more information on the performance of Rosetta in modeling and docking D3/GPCRs in general, please consult **Nguyen, E. D. et al. Assessment and challenges of ligand docking into comparative models of g-protein coupled receptors. PLoS One 8, (2013).**

Dopamine is an essential neurotransmitter that exhibits its effects through five subtypes of dopamine receptors, important members of class A G-protein coupled receptors (GPCRs). Both subtype two (D2R) and subtype three (D3R) function via inhibition of adenyl cyclase, and modulation of these two receptors has clinical applications in treating schizophrenia. However, the high degree of binding site conservation between D2R and D3R makes it difficult to generate pharmacological compounds that selectively bind one or the other, and thereby reducing side effects.

Today, we will examine how eticlopride, a D2R/D3R antagonist, binds to human D3R.

A crystal structure is available for the D3R and eticlopride complex (PDB: **3PBL**), but for the purposes of this exercise, we will model the protein-ligand interactions anyways. In reality, you may be using a comparative model rather than a crystal structure for the protein receptor, but the steps in this tutorial will apply to both.

For this exercise, we'll be doing our pre-docking preparations in the `protein_prep` and `ligand_prep` folders. The modeling will be done in the `docking` folder. The `scripts` folder contains helpful ligand docking specific scripts that we'll be using during this tutorial (*you should never be copying files to or from this folder*). All necessary files are also prepared in the answers directory in case you get stuck.

5.2 Step 1

Go to the desired location:

Navigate to the `ligand_docking` directory where you will find the `ligand_prep`, `protein_prep`, `docking`, and `answers` folders.

On macOS Terminal:

```
# macOS
cd $ROSETTA3_DEMOS/tutorials/ligand_docking/protein_prep
```

On Container Terminal:

```
#Container
cd /data/demos/tutorials/ligand_docking/protein_prep
```

5.2.1 Step 2: prepare human dopamine 3 receptor

Prepare a *human dopamine 3 receptor* structure. We will do this by obtaining the crystal structure (**3PBL**) and removing the excess information.

The first step has to be accomplished on the Docker Container side as the called python script `clean_pdb.py` calls on `wget` to download a PDB file but `wget` is not installed on macOS by default. The `clean_pdb.py` script then calls on `zcat` to unarchive the file. However, on macOS this software behaves differently and expect a file ending with `.Z` and cause a "file not found" error. Thus it is best to accomplish this task on the Container side, but since we are sharing the directories these will "magically" appear on the macOS side as well!

2.1. Change into the protein_prep directory with the `cd` command:

We can verify that we are already within the `protein_prep` directory from the last command above.

```
pwd
```

The original Tutorial suggested to download the PDB file, but the `clean_pdb.py` script will download it anyway so it is not really necessary.

2.2. Download `3BLP` (pdb format) from http://www.rcsb.org/pdb/home/home.do into the `protein_prep` directory.

The `clean_pdb.py` script will allow you to strip the PDB of information other than the desired protein coordinates. The '`A`' option tells the script to obtain chain A only. The full crystal structure consists of two monomers as a crystallization artifact.

On the Container Terminal type:

```
/data/tools/protein_tools/scripts/clean_pdb.py 3PBL.pdb A
```

2.3. There are two output files from `clean_pdb.py`: `3PBL_A.pdb` contains a single chain of the protein structure and `3PBL_A.fasta` contains the corresponding sequence. `3PBL_A.pdb` is the receptor structure we will be using for docking, copy this into the docking directory.

```
cp 3PBL_A.pdb ../docking
```

Note: This structure has a T4-lysozyme domain instead of the third cytoplasmic loop as a stabilizing feature for crystallography. Normally, we would truncate this lysozyme segment and perform loop modeling as discussed in the comparative modeling tutorial to regenerate the intracellular loop. However in the interest of time, we will use the lysozyme containing structure as the eticlopride binding site is far from the intracellular domain.

5.2.2 Step 3: prepare the ligand files

Next, we will prepare the ligand files by generating parameters using a eticlopride conformational library.

For more information about the ligand preparation, check ligand preparation tutroial| prepare_ligand.

3.1. `cd` into the directory named `ligand_prep`

```
cd ../ligand_prep
```

3.2.In the directory, you will find a pair of already prepared files: `eticlopride.sdf` and `eticlopride_conformers.sdf`

3.2.1. `eticlopride.sdf`: This contains the eticlopride structure found in the 3PBL protein complex.

Note: You can also find the ligand file from this particular PDB structure by going to the 3PBL page and scrolling down to the "Ligand Chemical Component" section. From there, you can click "Download" under the ETQ identifier.

3.2.2. `eticlopride_conformers.sdf`: This is a library of conformations for eticlopride generated outside of Rosetta. The downloaded ligand `.sdf` file only contains conformations found in the PDB so we must expand the library to properly sample the conformational space. We also need to add hydrogens since they are not resolved in the crystal structure. Feel free to open the file in PyMOL and use the arrow keys to scroll through the different conformations:

```
# Do not run
pymol eticlopride_conformers.sdf
```

The command `pymol eticlopride_conformers.sdf` assumes a Linux computer with a full Graphical interface and will not work on macOS or within the Docker Container as it is running as "Text-only."

To open PyMOL from command line on a Mac use the command:

```
open -a /Applications/PyMOL.app
```

Then slide the file `eticlopride_conformers.sdf` onto PyMOL with your mouse (using the file name as argument does not currently open it on macOS.) If you are not sure where the file is on your Mac, the following 2 commands will open the folder where it's located:

```
# Go to location:
cd ROSETTA3_DEMOS/tutorials/ligand_docking/ligand_prep

# Open current folder represented by a dot: .
open .
```

This particular conformational library was generated using the Meiler lab's BioChemical-Library (BCL). The BCL is a suite of tools for protein modeling, small molecule calculations, and machine learning. If you're interested in licensing the BCL, please visit http://www.meilerlab.org/bclcommons or ask one of the instructors.

Other methods of ligand conformer generation include OpenEye's MOE software, CSD Mercury software (CSD Conformer Generator) and web-servers such as Frog 2.1 or DG-AMMOS. The generated libraries will differ depending on the chosen method.

3.2.3. Generate a `.params` file and associated PDB conformations with Rosetta atom

types for eticlopride. A `.params` file is necessary for ligand docking because Rosetta does not have records for custom small molecules in its database.

This step is done using the script `molfile_to_params.py`.

Note: This command can be run from either Terminal... However, the first line of the script reads: `#!/usr/bin/env python` which assumes that the computer environment has a `python` path defined. On my Mac it is currently defined as `python3` and therefore it complains with *`env: python: No such file or directory`*. This is fixed easily by adding `python3` in front of the actual command on the macOS side.

You can run this step with either of the following commands. The first one could also work on Mac ***if*** `python` is defined as such. For the Mac Terminal option I make use of the environment variable `$ROSETTA3`. You can use the option `-h` first as suggested in the tutorial to understand the meaning of the command:

Option 1 (In Container):

```
# Show help
/data/source/scripts/python/public/molfile_to_params.py -h

# Run
/data/source/scripts/python/public/molfile_to_params.py -n ETQ -p ETQ --conformers-in-on
```

Option 2 (Mac Terminal):

```
# Show help:
python3 $ROSETTA3/scripts/python/public/molfile_to_params.py

# Run
python3 $ROSETTA3/scripts/python/public/molfile_to_params.py -n ETQ -p ETQ --conformers-
```

Note: You may encounter a warning about the number of atoms in the residue. This is okay as Rosetta is merely telling you that the ligand has more atoms than an amino acid.

File `ETQ.params` contains the necessary information for Rosetta to process the ligand, `ETQ.pdb` contains the first conformation, and `ETQ_conformers.pdb` contains the rest of the conformational library.

Note: The tutorial assumes that we are within the `ligand_prep/` directory, but also calls the `sdf` file with `ligand_prep/eticlopride_conformers.sdf` which will cause an error since we are *already within* that directory. Thus the directory name has been removed from the above commands.

3.2.4. If you use the `tail` command on `ETQ.params`, you will notice the `PDB_ROTAMERS` property line that tells Rosetta where to find the conformational library. Make sure this line has `ETQ_conformers.pdb` as the property.

```
tail ETQ.params
```

Note: The same command could be given on the macOS side as well.

3.2.5. Now that we have the necessary files for ligand docking, let's copy them over the `ligand_docking` directory.

```
cp ETQ* ../
```

Note: The same command could be given on the macOS side as well. Remember that bot Terminal are "looking within" the exact same directory.

5.2.3 Step 4: Final preparations in the docking directory

Now we want to make our final preparations in the docking directory.

4.1. Switch over to our `ligand_docking` directory

```
cd ../
```

4.2. Open up our prepared receptor and ligand structures to examine the complex

```
# Do not run
pymol 3PBL_A.pdb ETQ.pdb
```

The command `pymol 3PBL_A.pdb ETQ.pdb` invites to explore the complex graphically. Use the methods described previously for graphical exploration.

(Original NOTE: if you cannot open `pymol` from the command line, you may need to set up your `bash` environment.)

Now make a pdb file by concatenating your protein and ligand, running this script:

```
cp protein_prep/3PBL_A.pdb .
cat 3PBL_A.pdb ETQ.pdb > 3PBL_ETQ.pdb
```

If you don't have these files, copy them from the `answers` directory:

```
cp answers/docking/3PBL_ETQ.pdb .
```

4.3. Tip: 'All->Action->preset->ligand sites->cartoon' will help you visualize the protein/ligand interface. The All button is denoted by a single letter "A" in Pymol GUI.

Since this is a rudimentary exercise, we will start with the ligand in the protein binding site.

In practical application, we may need to define a starting point with the StartFrom mover or to manually place the ligand into an approximate region using PyMOL.

4.4. Once you close PyMOL, make sure Rosetta has these four necessary input structure/parameter files in the `ligand_docking` tutorial directory. If you are missing any of these, copy them from `../answers/docking/`

- `3PBL_A.pdb`: a single chain of the protein receptor structure

- `ETQ.pdb`: a default starting conformation for eticlopride
- `ETQ_conformers.pdb`: A pdb file containing all conformers generated from the eticlopride library.
- `ETQ.params`: a Rosetta parameter file that provides the necessary properties for Rosetta to treat eticlopride

5.2.4 Step 5: Rosetta wrapper and helpers

Next we need to make sure we have the proper RosettaScripts|rosetta_scripting XML file, input options file, and crystal complex (correct answer). These files are provided to you as `dock.xml`, `options.txt`, and `crystal_complex.pdb`. Copy these to your `ligand_docking` directory:

```
# Remember that . is the current dir
cp docking/dock.xml .
cp docking/options .
cp docking/crystal_complex.pdb .
```

- `dock.xml` - This is the RosettaScripts XML file that tells Rosetta the type of sampling and scoring to do. It defines the scoring function and provides parameters for both low-resolution coarse sampling and high-resolution Monte Carlo sampling.
- `options.txt` - This is the options file that tells Rosetta where to locate our input PDB structures and ligand parameters. It also directs Rosetta to the proper XML file.
- `crystal_complex.pdb` - This is the D3-eticlopride complex from the PDB. It will serve as the correct answer in our case allowing us to make comparisons between our models and actual structures.

5.2.5 Step 6: Run the docking study

Note: This is the point where using macOS binaries would prove valuable for larger computations and when it is useful to have the macOS binaries installed. For large projects the Mac binaries will run faster than running those within the Docker Container by emulation which would mean a double emulation on an M Series Silicon Mac: one to convert the Linux code, and the second to convert from Intel (amd64) to Silicon Chip (arm64), which is done seemlesly for Mac users by a utility aptly called "Rosetta2" but has not connection with the Rosetta we are using for protein or docking computations!

IMPORTANT NOTE: The name of the binary will differ depending on the operating system.

Run the docking study (This should take a few minutes at most, as we're using a reduced number of output structures):

The command in the original tutorial assumes a standard installation, with binary:

```
$> $ROSETTA3/bin/rosetta_scripts.linuxgccrelease @options
```

However, within the Docker Container the binaries have a different naming convention. The binaries within the container are within `/usr/local/bin` and the specific one to call for this section is named:

```
rosetta_scripts.cxx11threadserialization.linuxgccrelease
```

On the Mac it will be:

```
$ROSETTA3/bin/rosetta_scripts.static.macosclangrelease
```

To run the docking use the appropriate binary, followed by **`@options`**

On the Mac it would be:

```
$ROSETTA3/bin/rosetta_scripts.static.macosclangrelease @options
```

5.2.6 Step 7: Rosetta models

The Rosetta models are saved with the prefix `3PBL_ETQ_` followed by a four digit identifier such as `3PBL_ETQ_0001.pdb`. Each model PDB contains the coordinates and Rosetta score corresponding to that model. In addition, the model scores are summarized in table format in the `score.sc` file. The two main scoring terms to consider are:

- `total_score`: the total score is reflective of the entire protein-ligand complex and is good as an overall model assessment

- `interface_delta_X`: the interface score is the difference between the bound protein-ligand complex and the unbound protein-ligand. Interface score is useful for analyzing ligand effects and for comparing different complexes.

5.2.7 Step 8: Transform_accept_ratio

One other metric to keep an eye on is the `Transform_accept_ratio`. This is the fraction of Monte Carlo moves that were accepted during the low resolution Transform grid search. If this number is zero or very low, the search space may be too restrictive to allow for proper sampling.

5.2.8 STEP 9: RMSD

In benchmarking examples when we have a correct crystal structure, `ligand_rms_no_super_X` will give us the RMSD difference between our model ligand and the crystal structure ligand given in `crystal_complex.pdb`.

This is an important metric when benchmarking how well your models correlate to reality. When the crystal structure is unknown, we can also calculate model RMSDs using the best scoring structure as the "true answer".

5.2.9 Step 10: Use PyMOL for visual comparison

Use PyMOL to visually compare your best-scoring model and worse-scoring model with the crystal structure provided in `crystal_complex.pdb`.

The "All->Action->preset->ligand sites->cartoon" setting in PyMOL is ideal for visualizing interfaces.

What interactions were successfully predicted by Rosetta?

5.2.10 Step 11: Quick visusalization script

The `visualize_ligand.py` script in the `scripts` directory is a useful shortcut to doing quick visualizations of protein-ligand interfaces.

It takes in a PDB and generates a `.pse` PyMOL session by applying common visualization settings. The example below shows the command lines for using this script on the `0001` model but you are free to try it on any one (or more!) of your models.

This can to be done on the Docker Container side if the `pymol` Python package was installed (see above.) Therefore this is a step that cannot be run "as is" on macOS.

Note that the tutorial file name is `3PBL_A_ETQ_0001.pdb` but our result files do not have the `_A_` portion.

Run this command from the Docker Container Terminal, assuming we are within the `ligand_docking` directory which in turn contains the `scripts` directory:

```
scripts/visualize_ligand.py 3PBL_ETQ_0001.pdb
```

On the Mac side, double click the resulting `3PBL_ETQ_0001.pse` file which will open with PyMOL.

5.3 Analysis

Since we generated such a small number of structures, it is unlikely to capture all the possible binding modes that you would expect to encounter in an actual docking run. In the `out` directory, there are 50 models pre-generated using the exact same protocol.

We will look at an example of how we can analyze this dataset.

5.4 Analysis step 1:

`cd` into the out directory in your `ligand_docking`:

```
cd out
```

5.5 Analysis step 2:

Directory `out` also contains additional files: `score.sc`, `score_vs_rmsd.csv`, `rmsds_to_best_model.data`, and several `.png` image files.

- `score.sc`: summary score file for the 50 structures as outputted by Rosetta.

- `score_vs_rmsd.csv`: a comma separated file with the filename in the first column, total_score for the complex in the second column, the interface score in the third column, and ligand RMSD to the native structure in the fourth column.

This file was tabulated using the `extract_scores.bash` script and the `score.sc` file as input.

This is a very specific script made for extracting useful information in ligand docking experiments. However, the script can be easily customized for extracting other information from Rosetta score files. If you have any in-depth questions about how it works or how to modify it, feel free to ask.

To see how it in action, run:

```
# assumes we are in the out directory
../scripts/extract_scores.bash score.sc
```

- `rmsds_to_best_model.data`: a space separated file containing RMSD comparisons with the best scoring model (not crystal structure!) for all PDB files.

A more detailed discussion of this file will come further down in the tutorial. This file has the filename in the first column, an all heavy-atom RMSD in the second column, a ligand only RMSD without superimposition in the third column, a ligand only RMSD with superimposition in the fourth column, and heavy atom RMSDs of side-chains around the ligand in the fifth column.

This file is generated using the `calculate_ligand_rmsd.py` script. It uses `pymol` to compare PDB structures containing the same residues and ligand atoms. (As such this command does not run by default on macOS.)

It's a quick way of calculate ligand RMSDs of Rosetta models.

To see how this works, let's try it on the five models we generated in the previous steps:

STOP

The script `calculate_ligand_rmsd.py` on lines 83 and 221 using the `print` statement from Python version 2 that will cause an error. These 2 lines need to be converted to the `print()` function of Python 3. This can be done with the following stream editor (`sed`) regular expression script which edit and overwrite the original script:

```
# Assumes we are still in out directory
sed -i -r 's/^(\s*print)\s+(.*)/\1(\2)/g' ../scripts/calculate_ligand_rmsd.py
```

Note: if there is an error such as *sed: couldn't open temporary file: Permission denied* remove `-i` and create a new file with *e.g.* `.py3` extension:

```
# Assumes we are still in out directory
sed -r 's/^(\s*print)\s+(.*)/\1(\2)/g' ../scripts/calculate_ligand_rmsd.py > ../scripts/
```

```
# make it executable
chmod a+x ../scripts/calculate_ligand_rmsd.py3
```

(*Credit*: Copilot converted the `vi` editor command found on StackOverflow.) See the Appendix of my Blog for details.

Back to the Tutorial material:

```
# Change directory up one level
cd ../

# Run the script with .py or .py3 ending (see above)
./scripts/calculate_ligand_rmsd.py -n 3PBL_ETQ_0003.pdb -c X -a 7 -o rmsds_to_best_model
```

Run the script with `-h` to obtain relevant information that will detail the chosen options. For example `-c X` is for choosing chain `X`.

This command compares all five of your models to the one after the `-n` option. Your best scoring model may not be the one labelled 0003 so feel free to customize that option. The -c tells the script that the ligand is denoted as chain X. The -a tells the script to use 7 angstroms as the cutoff sphere for side-chain RMSDs. The -o option is the output file name. Lastly, we provided a list of PDBs using the wildcard selection.

The script produces the `rmsd_to_best_model.data` file that you can open in any text editor. Feel free to ask questions if you like to discuss more of how to customize this script for your own applications.

Now let's go back to the pre-generated model directory:

```
cd out
```

- `PNG files`: plots made from the various data file mentioned above. Python and the `matplotlib` package was used here but you are free to use any plotting software you prefer.

5.6 Analysis step 3: RMSD plots

In this case, we have the correct answer based on the crystal structure so we can examine a score *vs* rmsd plot to see if the better scoring models are indeed closer to the native ligand binding mode.

Open up the plot with the following command:

Use macOS command below or choose graphically.

```
# gthumb score_vs_crystal_rmsd_plot.png <- tutorial orginal command
# cd to out directory
cd $ROSETTA3_DEMOS/tutorials/ligand_docking/out
# Open file
open -a /System/Applications/Preview.app score_vs_crystal_rmsd_plot.png
```

On the X-axis you will see the ligand RMSD to the ligand in the crystal structure. On the Y-axis you will see the interface delta score in Rosetta Energy Units. Notice the general correlation between RMSD and Rosetta Score, with a large cluster of highly accurate and good scoring models in the lower left hand corner.

5.6.1 Analysis step 4: Best model

In practical applications, we would not have the crystal structure for comparison. However, we can treat the best scoring model as the correct model and see if we generate a similar funnel.

This is one application of how we might use the `calculate_ligand_rmsd.py` script discussed earlier.

Once we identify a desired "best model", we can run the script to generate the `rmsds_to_best_model.data`.

Some scripting may be required to put the information from multiple files together, depending on which software package you choose to graph with. To identify the best scoring model for this example, I selected the top **200** models based on the best overall score and then identified the best model by interface score.

The best model for these plots is `3PBL_A_ETQ_0347.pdb`. Open up the first plot with:

```
# gthumb score_vs_low_rmsd_plot.png <- tutorial original command
# macOS:
open -a /System/Applications/Preview.app score_vs_low_rmsd_plot.png
```

Again, we see a cluster of good scoring models near the best scoring model with a general downward trend further away. We can zoom in on the cluster in the lower left hand corner to get an even better picture.

```
# gthumb score_vs_low_rmsd_zoom_plot.png <- tutorial original command
# macOS:
open -a /System/Applications/Preview.app score_vs_low_rmsd_zoom_plot.png
```

We see the same overall trend in this cluster, suggesting that the top scoring models in this run are likely to be good predictors of the true ligand binding position.

5.6.2 Analysis step 5: structure visualizations

Finally let's look at some structures. To sort the CSV file by interface score and take the top twenty, type:

```
sort -t, -nk3 score_vs_rmsd.csv | head -n 20
```

Note: `-t,` defines the column/field delimiter as comma (since we are using a CSV file.) `-n` sorts numerically, on the 3rd column: `-k3`

These should all be very low RMSD models. To compare a certain structure to the native in PyMOL, use:

```
# Do not run
# pymol 3PBL_ETQ_0001.pdb ../crystal_complex.pdb <- tutorial original command
```

Open PyMOL on your Mac with `open -a` command shown previously.

Don't forget the ligand site preset mode for visualizing interfaces or use the `visualize_ligand.py` script to generate PyMOL session `.pse` files. If you like, we can also look at some of the poor scoring models to see exactly what went wrong. To find the *top 20 worse models* by interface score:

```
sort -t, -nk3 score_vs_rmsd.csv | tail -n 20
```

`3PBL_A_ETQ_0033.pdb` (or `3PBL_ETQ_0033.pdb`) should come up as a poor scoring, high RMSD structure. When we open it up in PyMOL, we can see that the ligand binding direction is different from the native position. This can happen when there is an extended binding pocket but in this case, the Rosetta score was able to discern the difference between these models.

```
# Do not run
# pymol 3PBL_A_ETQ_0033.pdb ../crystal_complex.pdb
```

Open PyMOL on your Mac with `open -a` command shown previously.

5.7 Notes on biding pockets

Some notes on binding pockets and adjusting the sampling space of the ligand

Q: My protein has a quite large cavity and a small ligand (not bigger than a Leucine). In the XML file these are the standard parameters: `<Transform name="transform" chain="X" box_size="7.0" move_distance="0.2" angle="20" cycles="500" repeats="1" temperature="5"/>`. Why is the `move_distance` and the angle so small? Would it make sense to increase the `box_size` to 12, the `move_distance` to 5 and the angle to 360 to sample more space the ligand is allowed to move in?

A: The Transform algorithm is a Monte Carlo procedure, and the `move_distance` and angle are the size for the individual steps in the MC protocol, not the maximum extent of the movement. They're also the `sd` of a *gaussian*, so you're not necessarily limited to the given amount in any given step. That said, if you're increasing the size of the pocket, it might make sense to bump the move size up in proportion. (So for a box size of 12, a `move_distance` of 0.25 to 0.5 or so might be appropriate.)

If you're exploring the pocket, I'd also suggest setting the `initial_perturb` option. By default Transform will always start with the input position. If you add the `initial_perturb=X.X` option, then it will first randomize the starting location of the ligand within an `X.X` Angstrom sphere from the starting position, as well as randomizing the orientation. – And a new random position/orientation will be taken for each nstruct,

so you can sample the pocket even if your MC moves aren't sufficient to wander across it. Also, if you're increasing the size of the pocket, you're likely going to need to increase the size of the Grid such that it will cover the maximal extent of ligand travel. If it doesn't, Transform will reject any ligand which accidentally falls outside the grid.

Q: Does the `initial_perturb` option also randomize the angles? Because there is another option `initial_angle_perturb` in the `TransformMover`.

A: Yes, by default setting `initial_perturb` will completely randomize the angles (ligand orientation) - the `initial_angle_perturb` is there if you want to reduce the angle perturbation. (*e.g.* if you're refining the orientation)

Congratulations, you have performed RosettaLigand docking study! Now use your docked models to generate hypotheses and test them in the wet lab!

6 Appendix

The `out` directory (see above) PNG plots of comparison of models with the crystal structure. There is no code explaining the creation of the plots and the text suggests that the plots were derived from "top **200** models" *i.e.* more than the 50 structures included in the directory. However, we could still plot results from these 50 structures with either `R` or `python`.

The following code was created by the Microsoft Copilot AI and is minimally edited for the name and location of the input file(s). It was tested and works!

6.1 R code

Assumes ggpplot2 has been installed, or install with command at the R console: `install.packages("ggplot2")`.

Make sure that you are in the directory containing the desired CSV file, or pointing to the correct directory (here `./out`). The size and color of the points within the plot was also added afterwards.

The plot is shown graphically but can be saved manually.

```
# Read the data from a text file (assuming the file is named 'data.txt')
data <- read.csv("out/score_vs_rmsd.csv", header=FALSE)

# Extract the 3rd and 4th columns
x <- data$V3
y <- data$V4

# Load the ggplot2 library for plotting
library(ggplot2)

# Create a scatter plot
ggplot(data, aes(x=x, y=y)) +
```

```
  geom_point(colour = "red", size = 3) +
  theme_minimal() +
  labs(title="Scatter Plot of 3rd and 4th Columns",
       x="3rd Column",
       y="4th Column")
```

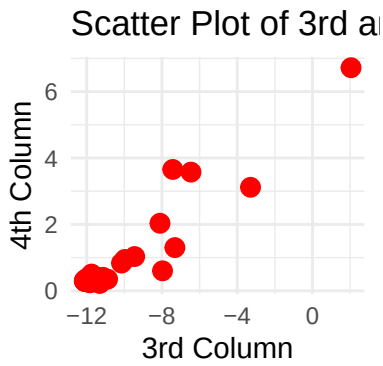

Figure 1: ggplot of score *vs* rmsd

6.2 Python 3 code

Assumes `pandas` and `matplotlib` are installed (*e.g.* using `pip` with the more modern command as: `python -m pip install pandas matplotlib`).

In python the first item is `0` hence 3rd and 4th columns are columns 2 and 3 (unlike in `R`.)

The script will export a PNG file called `plot.png`

```
import pandas as pd
import matplotlib.pyplot as plt

# Read the data from a text file (assuming the file is named 'data.txt')
data = pd.read_csv('./out/score_vs_rmsd.csv', sep=',', header=None)

# Extract the 3rd and 4th columns
x = data.iloc[:, 2]
y = data.iloc[:, 3]

# Create a scatter plot
plt.scatter(x, y)
plt.title('Scatter Plot of 3rd and 4th Columns')
plt.xlabel('3rd Column')
plt.ylabel('4th Column')
plt.grid(True)
```

```
# Save the plot to a PNG file
plt.savefig('plot.png')
```

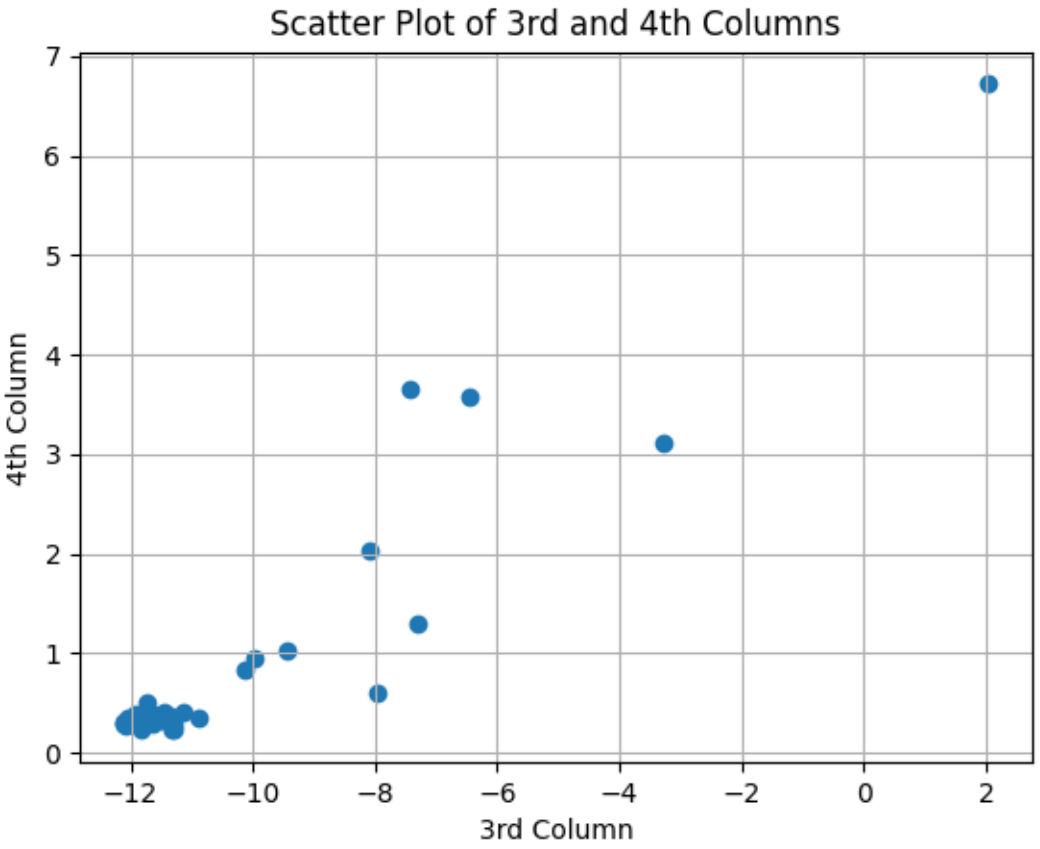

Figure 2: Python plot of score *vs* rmsd.csv