Workshop #2: PyRosetta

Rosetta is a suite of algorithms for biomolecular structure prediction and design. Rosetta is written in C++ and is available from www.rosettacommons.org. PyRosetta is a toolkit in the programming language Python, which encapsulates the Rosetta functionality by using the compiled C++ libraries. Python is an easy language to learn and also includes modern programming approaches such as objects. It can be used via scripts and also interactively as a command-line program, similar to MATLAB®.

The goals of this first workshop are (1) to have you learn to use PyRosetta both interactively and by writing programs and (2) to have you learn the PyRosetta functions to access and manipulate properties of protein structure.

Basic Elements

You will need a few basic tools to work on PyRosetta.

- You need a text editor to edit scripts. A good editor will "markup" your code in color and make sure your code is indented properly, and it can offer search tools across multiple files and sometimes support for running and debugging your program. One current favorite editor is **jEdit** (www.jedit.org). A popular editor on the Mac is **Aquamacs**, based on the program **Emacs**. **IDLE** is an "integrated development environment" (IDE) that is packaged with Python and includes pop-up function signatures while you are writing code. A text-only (no mouse) program is **vi** or **Vim** (www.vim.org), popular among *nix hackers. jEdit, Emacs, and vi are available for Windows and Linux platforms. There is a built in Mac editor called **TextEdit**, similar to **Notepad** or **WordPad** on a PC. These will not have the color markup and other tools, but they will allow you to edit your files. Choose one of these programs and learn to access it on your computer.
- You need a **command-line interface (CLI)** or **terminal**. On a Windows PC, typing `cmd` under the Start menu will launch a Command Prompt which will support standard DOS commands: `dir`, `cd`, `copy`, `type`, `more`, *etc*. On the Mac, you can find a terminal in the menu on the bottom of the screen or by searching for **xterm**. The Mac terminal will support standard UNIX/Linux shell commands: `ls`, `cd`, `less`, `cp`, `mkdir`, `rm`, `grep`, `awk`, `sed`, `gnuplot`, *etc.* Search the Internet if you are not familiar with Linux shell or DOS commands.
- You can access Python using the command `ipython` or `python ipython.py` from the terminal. We use **IPython** (rather than Python) since it supports tab-completion which will help us find PyRosetta functions. On Windows, your install may include a desktop shortcut for `iPython PyRosetta Shell`: this shortcut will open a terminal and start IPython for you.

Basic Python

Basic Python programming will be useful but is beyond the scope of this workshop. Excellent introductory and reference material on the Python language is available at docs.python.org. A very brief reference is also found in Appendix A.

Basic PyRosetta

1. Open a terminal and start IPython. To load the PyRosetta library, type

   ```
   from rosetta import *
   rosetta.init()
   ```
   or simply `init()`

   The first line loads the Rosetta commands for use in the Python shell, and the second command loads the Rosetta database files. The first line may require a few seconds to load.

2. Many pdb files, like the one you opened in Workshop #1, have extraneous information and often do not conform to file standards. You may have to "clean" your pdb file before loading it into PyRosetta. You can do this through the command line interface (not from within IPython) by using either the `grep` command (UNIX) or the `findstr` command (DOS) to remove all lines that do not begin with `ATOM` in the pdb file. Alternatively, a method from the PyRosetta `toolbox` namespace, `cleanATOM` can be used to create "clean" pdb files:

   ```
   from toolbox import cleanATOM
   cleanATOM("1YY8.pdb")
   ```

   (This method will create a cleaned `1YY8.clean.pdb` file for you.)

   See Appendix C for details on these methods and specific examples of how to clean pdb files.

3. Load a protein from a "clean" pdb file. Use the 1YY8.pdb file of the antibody you looked at in Workshop #1. Put the file in your working directory or change to the directory in which the file is located using `cd` from within IPython. Load the file as follows:

   ```
   pose = pose_from_pdb("1YY8.clean.pdb")
   ```

   This creates a `Pose` object that you can now work with using a variety of methods.

   If you have not already downloaded the pdb file, you can create a pose directly from the protein database if you have a connection to the Internet:

   ```
   from toolbox import pose_from_rcsb
   pose = pose_from_rcsb("1YY8")
   ```

(This method will also create `1YY8.pdb` and `1YY8.clean.pdb` files for you)

4. Examine the protein using a variety of query functions:

```
print pose
print pose.sequence()
print "Protein has", pose.total_residue(), "residues."
print pose.residue(500).name()
```

What type of residue is residue 500? _____

Note, this is the 500th residue in the pdb file but not necessarily "residue number 500" in the protein. Many pdb files have multiple peptide chains. Sometimes the residue numbering follows a convention from a family of homologous proteins, and often several residues of the N-terminus do not show up in a crystal structure. Find out the chain and pdb residue number of residue 500: _____

```
print pose.pdb_info().chain(500)
print pose.pdb_info().number(500)
```

Lookup the Rosetta internal number for residue 100 of chain A:

```
print pose.pdb_info().pdb2pose('A', 100)
```

The converse command is:

```
print pose.pdb_info().pose2pdb(25)
```

Get and display the secondary structure of the pose using a `toolbox` method:

```
from toolbox import get_secstruct
get_secstruct(pose)
```

To demonstrate IPython's *tab-completion* feature, type in `print pose.seq` and hit the tab key. IPython should complete the keyword `sequence` for you. Type `pose.` and hit the tab key, and you should see a list of functions available for `Pose` objects.

While we are examining the advantages of IPython, try out the built-in help features by typing any one of the following:

```
Pose?
?Pose
help(Pose)
```

Each of these will give a brief description of the Pose class and its purpose. The last form will also give a list of function signatures for all the available functions within the class. These methods of accessing help should work on many of the PyRosetta objects.

Protein Geometry

5. Find the φ, ψ, and χ_1 dihedral angles of residue 5:

```
print pose.phi(5)
print pose.psi(5)
print pose.chi(1, 5)
```

6. Find the N–C_α and C_α–C bond lengths of residue 5. There are at least a couple ways to do this.

   First, store the unique atom identifier codes in variables:

```
R5N  = AtomID(1, 5)
R5CA = AtomID(2, 5)
R5C  = AtomID(3, 5)
```

   (This works because the atoms are listed in a consistent order in a pdb file.) Then, use these identifier codes to lookup bond lengths in the conformation object:

```
print pose.conformation().bond_length(R5N, R5CA)
print pose.conformation().bond_length(R5CA, R5C)
```

   For the second method, access the Cartesian coordinates and use the `Vector` class to find the norm of the displacement vector between the two atoms:

```
N_xyz = pose.residue(5).xyz("N")
CA_xyz = pose.residue(5).xyz("CA")
N_CA_vector = CA_xyz - N_xyz
print N_CA_vector.norm
```

   These bond lengths are actual, experimental bond lengths from the crystal structure. When Rosetta creates proteins *de novo*, it uses ideal values, similar to those from Engh & Huber (1991). Let's check how the actual bond lengths compare to Rosetta's ideal values. Find the Rosetta database directory on your computer (*e.g.*, `/usr/local/PyRosetta/rosetta_database`). Enter the subdirectory `chemical/residue_type_sets/fa_standard/residue_types` and, with your text editor, load the `param` file appropriate for residue 5. The `ICOOR_INTERNAL` lines give the internal coordinates for an ideal conformation, including the torsion angle, bond angle, and bond length needed to build each subsequent atom in the group.

7. Can you identify the N–C_α and C_α–C bond lengths? How do they compare? Bonus: how do they compare to Engh & Huber's numbers? If they differ, why?

8. Find the N–C_α–C bond angle in radians:

```
print pose.conformation().bond_angle(R5N, R5CA, R5C)
```

   What is this angle in degrees? _____

   Again, compare with the Rosetta database ideal value. What is the hybridization of the C_α atom? _____ What is the standard bond angle for such a hybridization? _____

Be aware that not all bond lengths and angles are accessible through the conformation object. The conformation object only contains a minimal subset of bond lengths and angles used in generating Cartesian coordinates. The vector objects provide a general way to measure angles, distances, and torsions between arbitrary atoms.

9. How could you also find the N–C_α–C bond angle using the vector dot product function, `v3 = v1.dot(v2)`? (Recall from vector calculus that the angle between any two displacement vectors $\overrightarrow{BA}$ and $\overrightarrow{BC}$ is $\arccos\frac{\overrightarrow{BA}\cdot\overrightarrow{BC}}{|\overrightarrow{BA}||\overrightarrow{BC}|}$.)

Manipulating Protein Geometry

10. We can also alter the geometry of the protein. Perform each of the following manipulations, and give the coordinates of the N atom of residue 6 afterward.

```
pose.set_phi(5, -60)
pose.set_psi(5, -43)
pose.set_chi(1, 5, 180)

pose.conformation().set_bond_length(R5N, R5CA, 1.5)
pose.conformation().set_bond_angle(R5N, R5CA, R5C,
                                   110./180.*3.14159)
```

    New coordinates of N atom of residue 6: (_____, _____, _____)

Remember that only some bond lengths and angles are available through the conformation object. Note that even though dihedral angles are set in degrees, the bond angle is set in radians! (To make the conversion between degrees and radians easier, you may wish to import Python's `math` module. See Appendix A for more information.)

Visualization and the PyMOL Mover

What if we wish to view the changes to geometry that we have made? We can "dump" the information in a pose object into a new pdb file with the method `pose.dump_pdb("filename.pdb")` and then open this pdb file in our favorite visualization software. However, constantly dumping output and loading new files into a visualizer can be cumbersome; thus, the visualization process was streamlined with the `PyMOL_Mover`, which provides a means for observing structural changes almost instantaneously in PyMOL as they are made in PyRosetta.

First, we must open PyMOL and run a script that will cause PyMOL to listen for instructions from the PyMOL mover. (Certain Windows installations will use a shortcut that automatically does this for you, or you can create a `.pymolrc` file in your home directory in Linux or Mac that runs the code for you):

```
cd <your_PyRosetta_install_path>
run PyMOLPyRosettaServer.py
```

Then, with PyRosetta, we must instantiate a `PyMOL_Mover` and then apply it to a pose any time we make a change:

```
pymol = PyMOL_Mover()
pymol.apply(pose)
```

11. Make some changes to the dihedral angles of a pose and apply the PyMOL mover to watch the effect of the new angles on the structure.

For more advanced PyMOL mover options, visit www.pyrosetta.org/pymol_mover-tutorial, or see Appendix A.

Programming

12. You can write programs in Python to accomplish more complicated tasks. Using your text editor, open a new file with extension `.py` (*e.g.*, `rama.py`). You can write your entire program here and then run it either from the command line by typing

    ```
    [linux]> python rama.py
    ```

    or from inside a Python shell by typing

    ```
    In [1]: run rama.py
    ```

Here is a sample program:

```
from rosetta import *
init()
p = pose_from_pdb("1ABC.pdb")

for i in range(1, p.total_residue() + 1):
    print i, "phi =", p.phi(i), "psi =", p.psi(i)
```

Note that we must always first import the Rosetta modules with the `import` command and initialize Rosetta with the `init()` command before loading a pose. (Appendix A contains a list of common Python commands and syntax.) Test that you can write and run a simple program from a file.

Programming Exercises

Submit your script file and your output.

1. *Torsion angle.* Use the vector objects to write a script to calculate torsion angles between four arbitrary atoms. This will require knowledge of vector calculus. Hint: You will need to calculate the normal vectors of the two planes of the dihedral angle.

2. *Ideal helix*. Write a program to create a 20-residue ideal helix by setting the φ and ψ angles to the typical values for an α-helix. To start, use `pose = pose_from_sequence("AAA", "fa_standard")` to create a new pose, except use 20 "A"s in the argument to create a 20-residue poly-alanine. Output your structure using `pose.dump_pdb("helix.pdb")`.

   View your new file in PyMOL to check your work. How can you be sure your structure is a proper α-helix? List three distinct structural characteristics that you can check.

3. *Ideal strand*. Write a program to create a 20-residue ideal β-strand by setting the φ and ψ angles to values in the middle of the β region of the Ramachandran plot.

   View your new file in PyMOL to check your work. How can you be sure your structure is a proper β-strand? List three distinct structural characteristics that you can check.

4. *Secondary structure propensities*. Write a program to calculate the propensity of each residue type to appear in a helix. Loop through all residues in a protein, and count each alanine that is in a helix, sheet, or loop according to some φ/ψ-based criteria. The propensity can then be calculated as $P_{\alpha\ \mathrm{Ala}} = \frac{N_{\alpha\ \mathrm{Ala}}}{N_{\alpha\ \mathrm{total}}}$, $P_{\beta\ \mathrm{Ala}} = \frac{N_{\beta\ \mathrm{Ala}}}{N_{\beta\ \mathrm{total}}}$, and $P_{L\ \mathrm{Ala}} = \frac{N_{L\ \mathrm{Ala}}}{N_{L\ \mathrm{total}}}$, where $N_{\alpha\ \mathrm{Ala}}$, $N_{\beta\ \mathrm{Ala}}$, and $N_{L\ \mathrm{Ala}}$, are the counts of alanine residues in helices, sheets, and loops, respectively, and $N_{\alpha\ \mathrm{total}}$, $N_{\beta\ \mathrm{total}}$, and $N_{L\ \mathrm{total}}$, are the counts of *all* residues in helices, sheets, and loops, respectively. (Note that terminal residues have different names in Rosetta than internal ones; *e.g.*, an N-terminal `ALA` has the name `ALA_p:NtermProteinFull`.)

Bonus level 1: Find propensities for all 20 amino acid types. This will be easier if you use a data structure (list, array, dictionary, map) to store the counts of the 20 types. Do the residues with the highest helical propensity match that given by Brandon & Tooze?

Bonus level 2: To get better statistics, collect your data by looping over a set of 10 pdb files. Better yet, use a set of files such as the PDBSelect set of representative chains (http://bioinfo.tg.fh-giessen.de/pdbselect; this may require considerable download and compute time.)

5. *Idealize a protein.* Write a program that sets all bond lengths and angles to their Engh & Huber ideal values. Test your program using a structure from the pdb. What happens to the resulting protein? Why?

6. *Cleaning pdb files.* Coordinate files in the Protein Data Bank are quite diverse, and many pdb files have variations in their format to accommodate peculiarities such as post-translationally modified residues or disordered regions where coordinates could not be determined for certain atoms. In addition, some pdb files simply do not conform to the file standards. When the pdb file departs from the standards, it is necessary to clean-up the pdb file before loading it into Rosetta. (See Appendix C.) For this exercise, examine PDB ID 1D4I (HIV-1 protease in complex with an inhibitor).

   What non-standard amino acid is present, and what is this amino acid? (Hint: examine the pdb file header or the web page summary.)

   Write a script that converts the non-standard amino acid to its unmodified form. (Hint: use UNIX commands `grep`, `awk`, or `sed` along with pipes. Note: It is also possible to directly use a modified amino acid by creating a parameter file to define that residue, but that is a more advanced topic!)

   answer: The following UNIX shell command will change ABA to ALA and change the HETATM keys to ATOM (enter as a single-line command!):

```
awk '{if ($1 == "HETATM" && $4 == "ABA")
                                    {gsub("HETATM","ATOM");
                                    gsub("ABA","ALA")};
                                    print}' 1D41.pdb | grep
                                    ^ATOM > 1D41.clean.pdb
```

References

1. R. A. Engh & R. Huber, "Accurate bond and angle parameters for X-ray protein structure refinement," *Acta. Cryst. A* **47**, 392–400 (1991).
2. J. Parsons *et al.*, "Practical conversion from torsion space to Cartesian space for *in silico* protein synthesis," *J. Comp. Chem.* **26**, 1063–1068 (2005).
3. Python help available at http://docs.python.org.