

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.

1. 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**. 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.
2. You need a **command-line terminal**. On a Windows PC, the `StartPyRosetta.vbs` script (or **PyRosetta** shortcut) will open a terminal. On the mac, you can find a terminal in the menu on the bottom of the screen or by searching for **xterm**. The terminal will support standard unix/Linux shell commands: `ls`, `cd`, `less`, `cp`, `mkdir`, `rm`, `grep`, `awk`, `sed`, `gnuplot`. See the Google if you are not familiar with Linux shell commands.
3. You can access Python using the command **ipython** from the terminal. We use **iPython** (rather than `python`) since it supports tab-completion which will help us find PyRosetta functions.

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.

Basic PyRosetta

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

```
from rosetta import *
rosetta.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. Load a protein from a PDB file. Download an x-ray crystal structure of the ras-like domain of CENTG3 from entry 3IHW in the Protein Data Bank (<http://www.rcsb.org>). Put the file in your working directory. Load the file as follows:

```
pose = Pose("3ihw.pdb")
```

Many PDB files 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 with: `grep "^ATOM" 3ihw.pdb > 3ihw.clean.pdb`. Then load `3ihw.clean.pdb` into the pose. If the load function still gives errors, you might use your text editor to open the PDB file and edit or remove the offending data lines.

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

```
print pose
print pose.sequence()
print "Protein has ",pose.total_residue()," residues"
print pose.residue(5).name()
```

What type of residue is residue 5? _____

Note that this is the 5th residue in the PDB file, but not necessary “residue number 5” in the protein. 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 5: _____

```
print pose.pdb_info().chain(5)
print pose.pdb_info().number(5)
```

This protein has one chain, labeled chain L. Lookup the Rosetta internal number for residue 100 of chain A:

```
print pose.pdb_info().pdb2pose("A",5)
print pose.pdb_info().pose2pdb(25)
```

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

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

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

5. Find the N-C $_{\alpha}$ and C $_{\alpha}$ -C bond lengths of residue 5. There are at least a couple ways to do this.

First, using the atom identifier codes to lookup bond lengths in the conformation object:

```
R5N = AtomID(1,5)  
R5CA = AtomID(2,5)  
R5C = AtomID(3,5)  
  
print pose.conformation().bond_length(R5N,R5CA)  
print pose.conformation().bond_length(R5CA,R5C)
```

Second, accessing the Cartesian coordinates and using the vector class to find the norm of the 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 (e.g. /usr/local/PyRosetta/minirosetta_database). With your text editor, enter subdirectory chemical/residue_type_sets/fa_standard/residue_types and load the param file

appropriate for residue 5. The `ICOOOR_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.

6. Can you identify the $N-C_\alpha$ and $C_\alpha-C$ bond lengths? How do they compare? Bonus: how do they compare to Engh & Huber's numbers? If they differ, why?
7. Find the $N-C_\alpha-C$ bond angle:

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

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.

8. How could you also find the $N-C_\alpha-C$ bond angle using the vector dot product function, `v3 = v1.dot(v2)`?

Manipulating protein geometry

9. 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)
```

Remember that only some bond lengths and angles are available through the conformation object. Note that even though dihedrals are set in degrees, the bond angle is set in radians!

Programming

10. We 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]: import rama.py
```

Here is a sample program:

```
from rosetta import *
rosetta.init()
p = Pose("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 first initialize Rosetta with the import command, and load the pose. Test that you can write and run a simple program from a file.

Programming exercises

Submit your script file and your output.

1. Use the vector objects to write a script to calculate torsion angles between four arbitrary atoms.
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 `make_pose_from_sequence(pose, "AAA", "fa_standard")`, 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 -sheet? 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 which is helical, sheet, or loop according to some ϕ/ψ based criteria. The propensity can then be calculated as $P_\alpha = \frac{N_\alpha}{N_T}$, $P_\beta = \frac{N_\beta}{N_T}$, and $P_L = \frac{N_L}{N_T}$, where N_α , N_β , N_L , and N_T are the counts of helical, sheet, loop, and total alanine residues, respectively.

Bonus level 1: Find propensities for all 20 amino acid types. This will be easier if you use a data structure like a list or array 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 which 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. For this exercise, examine PDB ID 1d4l (HIV-1 protease in complex with an inhibitor).
 - a. What non-standard amino-acid is present, and what is this amino acid? (Hint: examine the PDB file header or the web page summary.)
 - b. Write a script which converts the non-standard amino acid to its unmodified form. (Hint: use Linux 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}'
1d4l.pdb | grep ^ATOM > 1d4l.clean.pdb
```

References

1. R. A. Engh & R. Huber, "Accurate Bond and Angle Parameters for X-ray Protein Structure Refinement," *Acta. Cryst.* **A47**, 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://python.org/doc>.