Project 2: Protein Design

Part 3: Replacing the Ligand and Optimizing the Interface without Mutating the Bromodomain

Assigned: 11/15 Due: 11/27 at 11:59 a.m.

In this part of the project, we're going to use PyRosetta to replace the original peptide ligand with our ligand of interest and optimize the interface **without** introducing mutations into the bromodomain.

In addition to **Appendix B**, at the end of this document, the following readings from the PyRosetta website are highly recommended before you start working:

Workshop #2: PyRosetta: http://www.pyrosetta.org/tutorials#TOC-Workshop-2:-PyRosetta

Workshop #6: Packing & Design: http://www.pyrosetta.org/tutorials#TOC-Workshop-6:-Packing-Design

For Workshop #6, read both the PDF associated with the tutorial and examine '*the basic protocol'* component of the companion script: http://graylab.jhu.edu/pyrosetta/downloads/scripts/demos/D050_Packer_task.py.

1. Look in the file peptide_assignments.xls for the peptide sequence you've been assigned. You will use the same PDB file as in Part 1 (2DVQ). Make sure the PDB file is in the current python directory.
2. Use PyRosetta to replace the original peptide with your new peptide sequence. Allow for any rotamers in the new peptide. To do so, some of the commands listed at the end of Appendix B and in the Packing and Design workshop may be helpful.
   a. **Hint**: you may want to use the **generate_resfile_from_pose** function to generate a *resfile* file which you can edit in a text editor to mutate or repack the appropriate residues. You can identify the appropriate index associated with each residue by using PyMol, by opening the PDB file in a text editor, or using native PyRosetta commands.
   b. Optional: check out the section called **Visualization and the PyMOL Mover** in the Workshop #2.
3. Optimize the interface between the new ligand and the bromodomain without introducing mutations into the bromodomain. You should once again use PyRosetta's repacking functionality. Hint: If you're smart about it, you can combine steps 2 and 3 into a single step.
4. In PyRosetta, save the PDB for your new optimized complex using the **dump_pdb("Filename.pdb")** function associated with pose objects.
5. Visualize your new complex in PyMOL, following the guidelines outlined in step 3 of Part 1.

What to submit:

In a .zip file named **LASTNAME_FIRSTNAME_PART3.zip**, include the following files

1. PyRosetta script used to generate new pdb and any resfiles used in repacking
2. PDB file of complex (repligand.pdb file) after optimizing interface

3. Ray traced image of complex after optimizing interface(repligand.png file)
4. PDF report (part3.pdf, described below)

In a PDF report, include the ray traced images requested above. In addition, explain how you used PyRosetta to optimize your interface and please answer the following questions:

1. What PyRosetta functions did you use to complete this part of the project and why did you use these functions?
2. Which residues in the ligand did you allow to repack and why?
3. How did you identify the residues in the bromodomain that you allowed to repack? Why did you choose these residues?
4. What is the potential energy of the complex formed between the bromodomain and the new peptide?

This report must be in paragraph format and only complete sentences are allowed.

Appendix A: Basic PyMOL

PyMOL is a viewer for macromolecular structures, typically derived from x-ray crystallography or NMR experiments. These structures are typically obtained from the RCSB Protein Data Bank (www.pdb.org) and come in pdb format. The full PyMOL manual is available at http://pymol.sourceforge.net/newman/userman.pdf , and is relatively thorough. Most PyMOL commands are available either through drop-down menus or a command line. Here I'll typically address the way to do things via drop-down menus.

Let's start by obtaining a pdb file to play with. First download the pdb file of the bromodomain that was assigned to you (all pdb entries are named with a 4-character code). To do so, use the following instructions:

1. Open chrome in your nomachine client by clicking on the chrome icon in the lower left hand corner.
2. 5ownload the relevant PDB structure. This should place the file in your Downloads fold. You most likely want to move it into the directory /home/yournamehere. To do so, use the graphical file interface provided by the third button in the bottom left of your screen.
    a. Note that you can also play around with any other .pdb file. You can get them from http://www.rcsb.org/pdb/home/home.do for any structure in the database.

Open it with PyMOL. A mass of sticks should appear on the screen. You can change your viewpoint by left-clicking and dragging to rotate the object. If you are on a laptop, one-button mode is probably easiest for viewing. Select this from the mouse menu. If you have a 3-button mouse, select 3-button viewing. Note that 2-button viewing can be helpful if your mouse wheel is troublesome.

Here are some basic mouse actions (these are also noted in the lower left corner of the viewing window).

Desired Action	**1-button mode**	**3-button mode**
zoom in and out	control and drag up/down	Right click and drag
translate the object in the viewing plane	alt and drag	Center button and drag

PyMOL will show the sequence of the protein if you select 'sequence' under the Display menu on the command line window. You can then select particular residues by clicking on the letter in the sequence at the top of the viewer window. You can also select residues by simply clicking on them in the viewer. The atoms will have pink dots on them to indicate they are selected. You can modify the properties of whatever you have currently selected by using the drop down menus to the right of '(sele)' in the right column.

Another way to select things is via the command line. The syntax is select <name>, <category> <identifier>

<name> is what you want to name your new selection.

<category> can be any of:

resn - residue type
res - residue number
name - atom in amino acid: N, C, O, Ca, Cb, etc.
symbol - element name
chain - chain A, B, C, etc.
ss - secondary structure type: 'h', 's' or 'l'

Selection criteria (in the form of <category> <identifier>) can be linked together by boolean operators: **and**, **or**, and **not**. When you create a new selection, it will show up in the right hand panel and you can access the drop down menus to manipulate only the atoms in that selection. For example, type **select chainA, chain A.** This way, you can modify only chain A of the molecule. For example, on “all,” click on H and choose everything, then chainA, click S and “cartoon” to show the cartoon version of only chain A. In addition to creating a new selection, the above syntax for selecting parts of the structure is generally used throughout pymol. For example, **color red, res 40** will color residue 40 red.

Appendix B: Basic PyRosetta

Open a terminal window, and type **python**
To load the PyRosetta library, type

from rosetta import *
rosetta.init()

Load a protein from a PDB file (a cleaned pdb you made using grep as shown in Part 1 above)

pose = Pose()
pose_from_pdb(pose, "3ihw.clean.pdb")

Examine the protein using a variety of query functions:

print pose
print pose.sequence()
print pose.total_residue()
print pose.residue(5).name()

How many residues does this protein have? 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 A. Lookup the Rosetta internal number for residue 91 of chain A:

print pose.pdb_info().pdb2pose("A",91)
print pose.pdb_info().pose2pdb(25)

Protein geometry

Find the ϕ, ψ, and χ1 angles of residue 5:

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

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)

To see the Cartesian coordinates of the N atom, use the command:

pose.residue(5).xyz("N")

Using Score Functions

Score functions are Rosetta's way of calculating energies – similar to a ΔG (but not including an entropy term and energies are unitless). Here, we will explore the types of energy Rosetta considers, and use an energy function to score our protein.

```
scorefxn = create_score_function('standard')
```

The option for standard tells Rosetta to load the standard all-atom energy terms. To see these terms, you can print the score function:

```
print scorefxn
```

Set up your own custom score function which includes just van der Waals, solvation, and hydrogen bonding terms, all with weights of 1.0. Use the following to start:

```
scorefxn2=ScoreFunction()
scorefxn2.set_weight(fa_atr,1.0)
scorefxn2.set_weight(fa_rep,1.0)
```

Enter the other weights and then confirm that the weights are set correctly by printing your score function.
Evaluate the energy of pose with your score function

```
scorefxn2(pose)
```

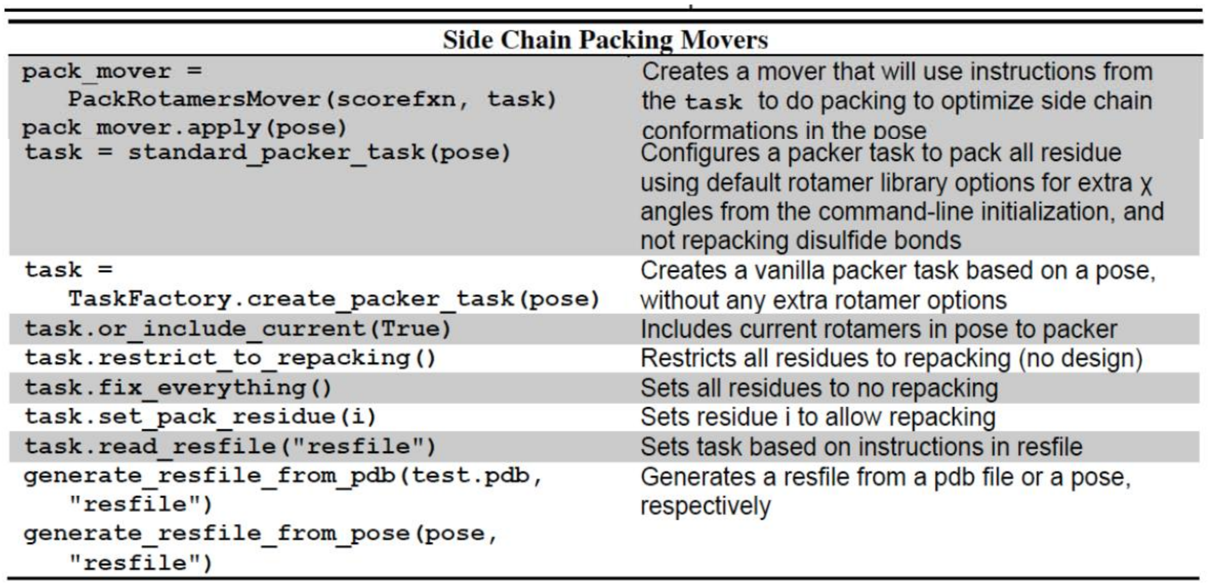
Evaluate the energy of pose with the standard score function:

```
scorefxn(pose)
```

Break the energy down into its individual pieces:

```
scorefxn.show(pose)
```

These represent a brief introduction to PyRosetta but you'll need to use more complex functions to complete this assignment. The PyRosetta tutorials (http://www.pyrosetta.org/tutorials) provide a really great reference and Workshop #'s 1, 2, 3 and 6 should be sufficient for you to complete this assignment.



Side Chain Packing Movers	

`pack_mover =`
`   PackRotamersMover(scorefxn, task)`
`pack_mover.apply(pose)`
	Creates a mover that will use instructions from the **task** to do packing to optimize side chain conformations in the pose

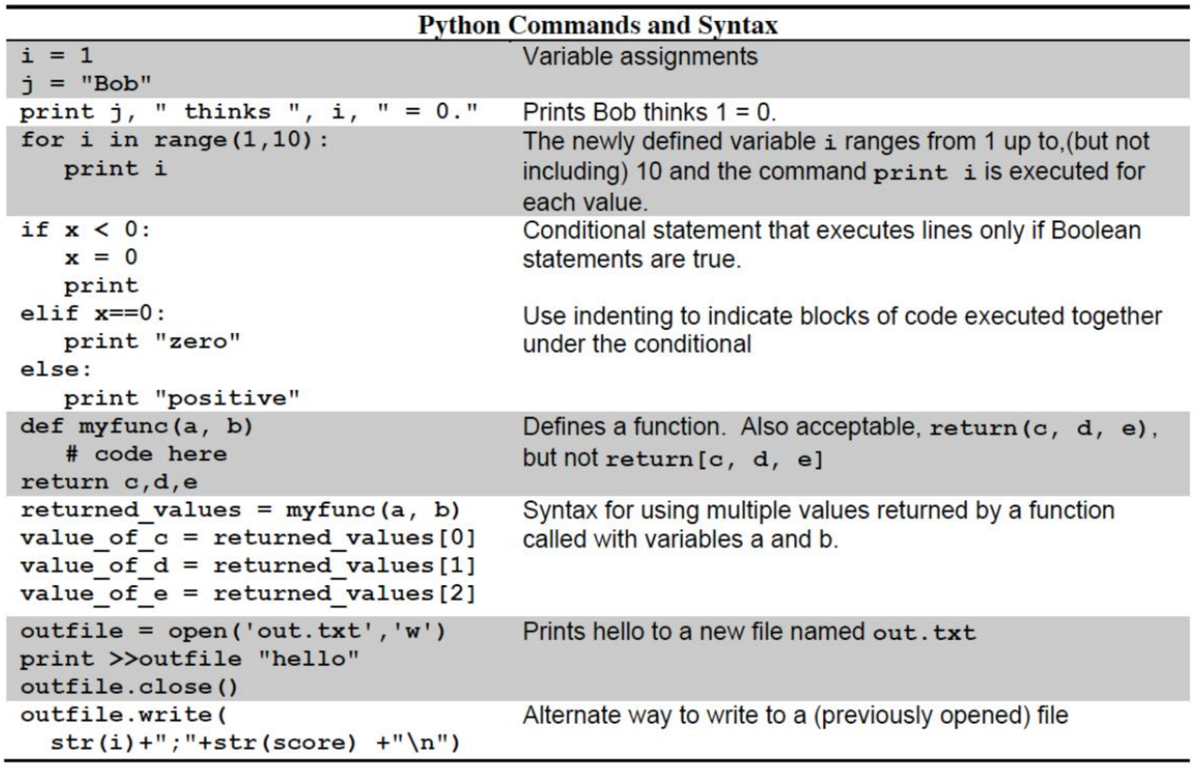
`task = standard_packer_task(pose)`	Configures a packer task to pack all residue using default rotamer library options for extra χ angles from the command-line initialization, and not repacking disulfide bonds

`task =`
`   TaskFactory.create_packer_task(pose)`
	Creates a vanilla packer task based on a pose, without any extra rotamer options

`task.or_include_current(True)`	Includes current rotamers in pose to packer
`task.restrict_to_repacking()`	Restricts all residues to repacking (no design)
`task.fix_everything()`	Sets all residues to no repacking
`task.set_pack_residue(i)`	Sets residue i to allow repacking
`task.read_resfile("resfile")`	Sets task based on instructions in resfile

`generate_resfile_from_pdb(test.pdb,`
`   "resfile")`
`generate_resfile_from_pose(pose,`
`   "resfile")`
	Generates a resfile from a pdb file or a pose, respectively

Here are some commands you might find useful:



Python Commands and Syntax	

`i = 1`
`j = "Bob"`
	Variable assignments

`print j, " thinks ", i, " = 0."`	Prints Bob thinks 1 = 0.

`for i in range(1,10):`
`   print i`
	The newly defined variable **i** ranges from 1 up to,(but not including) 10 and the command **print i** is executed for each value.

`if x < 0:`
`   x = 0`
`   print`
`elif x==0:`
`   print "zero"`
`else:`
`   print "positive"`
	Conditional statement that executes lines only if Boolean statements are true.
	
	Use indenting to indicate blocks of code executed together under the conditional

`def myfunc(a, b)`
`   # code here`
`return c,d,e`
	Defines a function. Also acceptable, **return(c, d, e)**, but not **return[c, d, e]**

`returned_values = myfunc(a, b)`
`value_of_c = returned_values[0]`
`value_of_d = returned_values[1]`
`value_of_e = returned_values[2]`
	Syntax for using multiple values returned by a function called with variables a and b.

`outfile = open('out.txt','w')`
`print >>outfile "hello"`
`outfile.close()`
	Prints hello to a new file named **out.txt**

`outfile.write(`
`  str(i)+";"+str(score) +"\n")`
	Alternate way to write to a (previously opened) file