

CS 483 Assignment 3

Due dates: Thurs. Feb. 16. Tues. Feb. 14 (to get 6 bonus marks).

Marks:

Exercise 1: [6]. Exercise 2: [2+6+1+1+1+1 = 12] Exercise 3: [8+2 = 10] Exercise 4: [7+8 = 15]

Exercise 5: [5+2+8+5 = 20] Total: **63**

The assignment will be marked out of **60**.

Exercise 1: A Useful Identity

Suppose we define the m by n matrix Q using n column vectors each with m components:

$$Q = \begin{bmatrix} q^{(1)} & q^{(2)} & \cdots & q^{(n)} \end{bmatrix}.$$

Show that, if v is a column vector with m components, then:

$$QQ^T v = \sum_{i=1}^n (q^{(i)T} v) q^{(i)}.$$

Exercise 2: Finding Cysteine Bridges

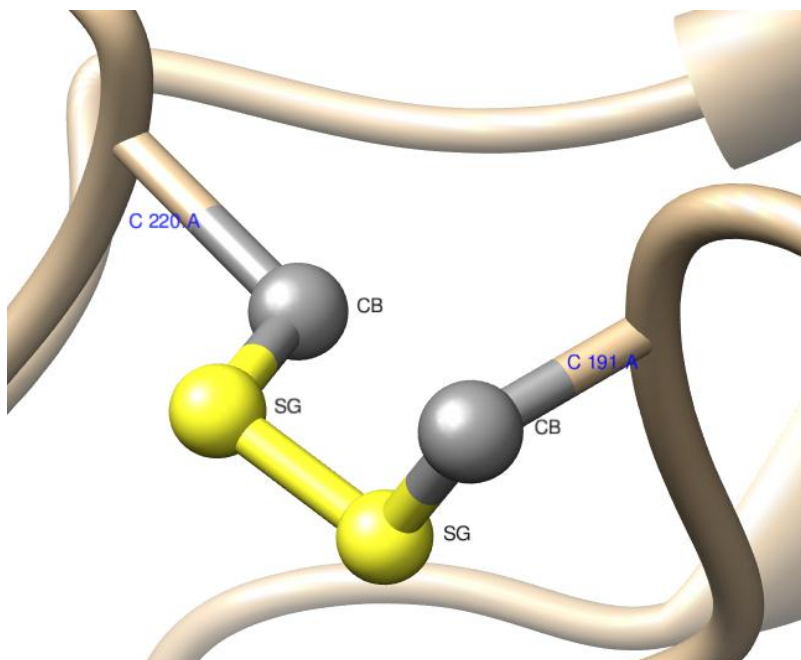
Recall what is meant by the protein folding “problem”. As a final step in the prediction of protein conformation, some algorithms will have predicted the conformation of the backbone but without specification of the disulfide bridges between cysteine residues. As a final step in the refinement of positions for the backbone atoms, the algorithm will then try to predict the occurrence of these bridges and will make adjustments to the backbone atoms so that the bridges can be put into the final predicted model. To do this, it is usually necessary to collect statistics that characterize the geometry of disulfide bridges.

The goal of this exercise is to write a Python script that can collect such information. Your script should fetch a protein file specified by user input typed into the Python shell. The user should also be given a choice of chain identifiers so that a chain can be designated by typing that identifier into the shell window. It should scan the backbone of the chain to find cysteine residues that are linked to other residues by means of a disulfide bridge. Be careful, not all cysteine residues are involved with bridges (for example, the D chain of 1A4Y has several cysteine residues that are not involved with disulfide bridges). In the next figure we can see a

disulfide bridge linking CYS 191.A with CYS 220.A in chain A of 1BQY. For each bridge found in the protein your script should print out the following information:

- i) The distance between the alpha carbon atoms in the cysteine residues involved in the bridge.
- ii) The dihedral angle determined by the four atoms (CB, SG, SG, CB) forming the bridge. (Use your notes to write a function that will compute such an angle.)
- iii) The residue names and positions of the cysteine residues that are involved (for example: CYS 191.A, and CYS 220.A).
- iv) The type of secondary structure containing each of the cysteine residues. In the figure both CYS residues are in coil secondary structures. Choices are 'coil', 'helix', or 'strand'.
- v) The names of residues before and after the cysteine residues.
For example: THR 190.A, HIS 192.A and PRO 219.A, GLY 221.A.
- vi) If a cysteine residue is in a backbone but does not participate in a disulfide bridge then it should be part of the output with a note indicating that there is no bridge structure associated with this residue.

Send well documented code to the TA along with output for each of the following: 1BQY (chain A), 1TCR (both chains), and 1WPD (all models)¹. In the last case, do parts iii) to v) only once since these entries will be same for all models in the file.



¹ Disulfide bridges in scorpion toxins: <http://www.jbc.org/content/early/2012/01/10/jbc.M111.329607.full.pdf>

Exercise 3: B-Factors and Protein Flexibility

The B-Factor of an atom is linearly related to the mean square displacement of an atom from its average position. Various research papers have used the B-Factor of an alpha carbon atom as a first approximation for the assessment of the flexibility of the protein backbone at that position: a larger B-Factor indicates more flexibility. Note that `bfactor` is an attribute for an atom in the Chimera object model.

A word of caution: The B-Factor is influenced, to some extent, by the effects of crystal packing prior to X-ray analysis. Consequently, the use of B-Factors does not necessarily give us a completely accurate assessment of the protein's flexibility as it would occur in the cytosolic environment².

The goal of this exercise is to assess the change in flexibility when HIV protease is co-crystallized with Tipranavir (one of the protease inhibitors currently available as an anti-HIV drug). The exercise will also give you some experience with the use of plotting facilities that are part of `matplotlib` module.

Steps for this assignment:

- 1) Get familiar with the plot facilities
- 2) Generate the data
- 3) Generate a bar chart for the data and make appropriate observations.

Consider the following script which builds a bar chart that compares the values stored in two arrays `data_A` and `data_B`.

```
import chimera
import matplotlib.pyplot as plot
import numpy as np
from numpy import *

data_A = np.array([2, 4, 6, 7, 3])
data_B = np.array([3, 5, 8, 6, 1])

width = 0.4
xVals = np.arange(1, 6)

plot.subplot(1,1,1)
plot.bar(xVals, data_A, width, color = 'g')
plot.bar(xVals + width, data_B, width, color = 'r')

plot.xticks(arange(0, 6, 1))

plot.ylabel("y-axis label")
plot.title("Title goes here")
plot.show()
```

² Song, G. and Jernigan, R.L., vGNM: A Better Model for Understanding the Dynamics of Proteins in Crystals. *J. Mol. Biol.* (2007) **369**, 880-893.

More details about plot facilities can be found at: <http://matplotlib.sourceforge.net/> .

To generate the data, your script should access two PDB files: 1HHP (the “apo” or ligand-free form of HIV protease) and 2O4P (HIV protease in complex with Tipranavir. For each protein, you should generate an array containing the B-Factors of the 99 alpha carbon atoms in chain A.

A direct comparison of the B-Factors corresponding to two different crystallizations is not advisable because each X-ray analysis may go through a refinement procedure that leads to different scales for the B-Factors³. So, each data set should be modified by a normalization calculation:

$$Bfactor = \frac{(Bf - \langle Bf \rangle)}{\sigma(Bf)}$$

where Bf is the B-Factor that Chimera gets from the PDB file, $\langle Bf \rangle$ is the mean B-Factor computed across all the alpha-carbon atoms of the protein chain (except for the first three and last three residues), and $\sigma(Bf)$ is the standard deviation computed across all the alpha-carbon atoms of the protein chain (except for the first three and last three residues). The first three and last three residues are usually omitted because chain termini are typically very flexible and may lead to bias in the computation of the mean and standard deviation values. The mean and standard deviation functions are available in the `numpy` module.

Use a modified version of the plot script to show how the two proteins differ in their B-Factors as you move along the 99 alpha carbon atoms in the backbone. (Use green for the apo form and red for the inhibited form of the protein).

Now answer the question: How is the information that you see in the bar chart consistent with the behaviour of the Tipranavir inhibitor acting as an anti-HIV drug? Hint: Residues 46 to 56 are usually considered to be the so-called “flap” region of the protease.

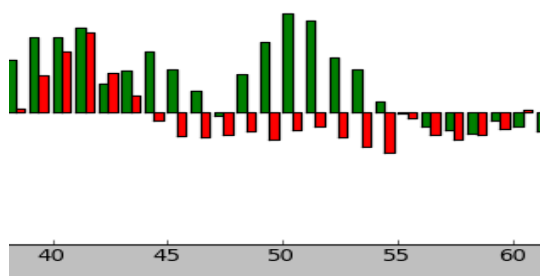


Fig. 1:A section of the plot comparing the B-Factors.

Send well documented code to the TA. Be sure your hand-in includes the answer for the last question.

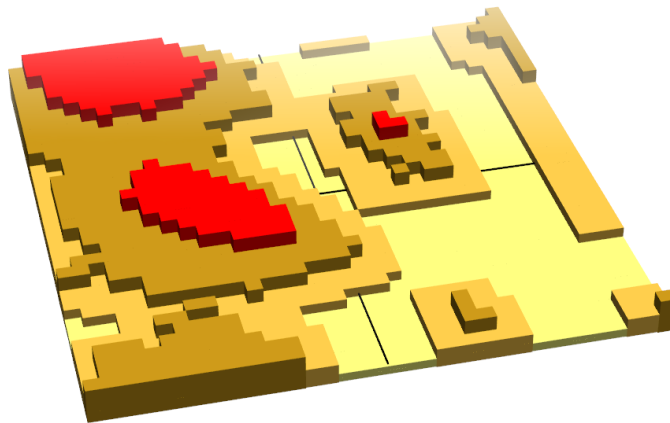
³ Yuan, Z., Zhao, J., and Wang, Z.X., Flexibility analysis of enzyme active sites by crystallographic temperature factors. *Protein Engineering*, vol. 16, no. 2, pp. 109-114, 2003.

Exercise 4: The Ramachandran Plot

Chimera allows you to place 3D objects into the scene. These can be drawn using a subset of the VRML (Virtual Reality Modeling Language). If you want extra information, see <http://en.wikipedia.org/wiki/VRML> for a quick overview.

This exercise involves the following steps:

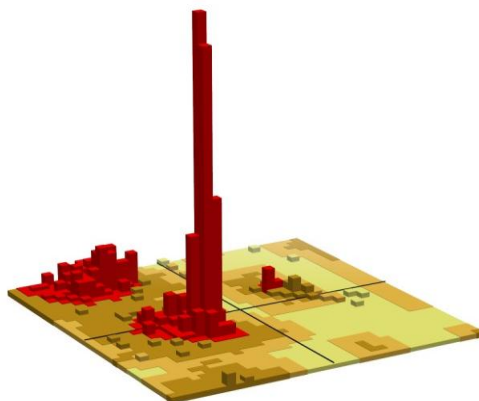
- a) Write a Python script that computes the (ϕ , ψ) pair of dihedral angles for all residues in a protein. Note: A pair can only be computed if the three backbone atoms of the residue are present along with the two adjacent backbone atoms in the residues before and after this residue. Your code should skip any residues that do not comply with this requirement. For example: the residue at the beginning or at the end of a chain will not qualify. The output of a computation should be dihedral angle pairs expressed in degrees. You should do the calculations “from scratch” using the formulae presented in class. It is possible to compute ϕ and ψ angles using methods that come with Chimera, but you are only allowed to use these functions if you wish to verify that your script is operating correctly.
- b) To provide a visualization of the pair data collected, you are to display the results in a 3D Ramachandran plot that is drawn with the VRML subset. To reduce your work load the course website has a Python script that contains all the needed VRML code. You should start by running my script to verify that it works. After tilting, the display should look like the following:



Note that this display is simply the background of a Ramachandran plot (as seen in colour Figure 7.7 of your text). If you inspect the Python starter script, you will see a large array (a list of lists) containing all the baseColorCodes. In the plot produced by the starter script, we have the height of each “pixel tower” being simply determined by the colour code. This is accomplished by passing the `testCaseCS483` table to the `vr_plot` function. Your mission in this part of the assignment is to replace the `testCaseCS483` table definition with code that will calculate all the (ϕ , ψ) pairs for a protein and then display the results using the plot function. For example, processing the protein with PDB ID 2FIF should give a plot that looks like the next figure.

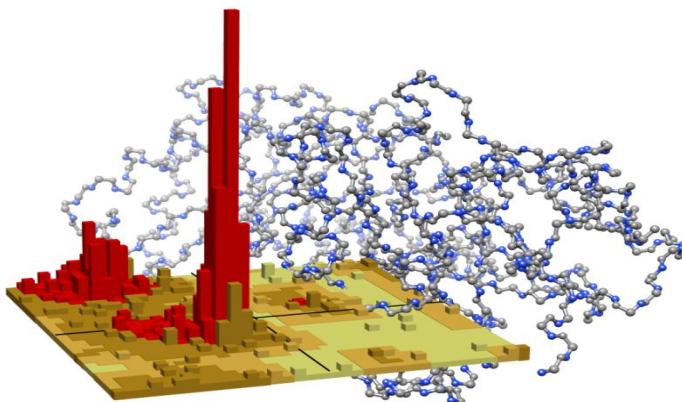
You can imagine that the input array is a set of “buckets” that contain a count of the number of times some (phi, psi) pair falls into a particular range of values. For example, the bucket entry in row 0 and column 1 will be incremented each time the (phi, psi) is such that $-170 \leq \phi < -160$ and $170 \leq \psi < 180$.

Once all residues make their contributions, the plot can be displayed by passing the bucket list to `vr_plot`.



For this exercise, you should hand in all scripts including session files. Test your program on 2FIF and 1A4Y. Note that your results for 1A4Y should be consistent with colour Figure 7.8 of the text.

NOTE: Both models (protein and plot) will usually appear on the display but you want to eliminate the protein model and show only the 3D plot. There is probably a way to control this from the Python script (I am still investigating this). Use the Favorites → Model Panel to determine the models that are to be Shown and/or Active (for zooming and rotating). The Focus and Set Pivot items in the Action menu can also help you get a good display and, of course, there is always the View All feature under the Side View tab in the Viewing panel. Combining model displays is artistic but not useful:



Exercise 5: Protein Animation

The object of this last exercise is to make an animation that gives you information about protein flexibility and movement in a thermal environment such as a living cell. You can find on YouTube several short videos that show such animations with various levels of sophistication:

- i) <http://www.youtube.com/watch?NR=1&feature=endscreen&v=ZCrJtNvoWf4> showing small movements of secondary structure
- ii) <http://www.youtube.com/watch?v=meNEUTn9Atg&feature=related> showing a computationally generated simulation of protein folding
- iii) <http://www.xvivo.net/the-inner-life-of-the-cell/> showing multi-molecular interactions within a cellular environment.

For more advanced animations look at the higher resolution 90 MB file that can be downloaded from: <http://www.xvivo.net/news/demo-kit/>. This shows a range of structures from the molecular to the tissue level.

The animations for this exercise will be more like that seen in (i) and we will not require any background music. ☺

The script will be using NMR generated data as input. For example: 1BMR which contains 25 different models indexed from 0 to 24 but named as #0.1, #0.2, ..., #0.25. You should go through the following steps:

- a) Write a function called `rmsdForModels(ix, jx)` which accepts two integers designating two different models within the same protein file. The output should be the RMSD for the two proteins (see below).
- b) Write a function called `buildDissimilarityArray()` that generates a symmetric hollow array such that the entry in row `ix` and column `jx` is given by the value computed as `rmsdForModels(ix, jx)`. By “hollow” we mean that all entries on the main diagonal are 0.
- c) Write a function called `buildAnimation(ds)` that accepts the dissimilarity array as input and then constructs an animation list (see below).
- d) Write a function called `doAnimation(ani_List, numCycles)` that will generate the animation (see below).

RMSD

Suppose index `ix` designates a model that we will refer to as protein *P* while index `jx` designates a model that we will refer to as protein *Q*. We calculate the RMSD using the formula:

$$RMSD(P, Q) = \sqrt{\frac{1}{n} \sum_{k=1}^n \|p_{\alpha}^{(k)} - q_{\alpha}^{(k)}\|^2}$$

where: $p_{\alpha}^{(k)}$ represents the coordinates of the alpha carbon for the residue with position number k found within protein *P* and $q_{\alpha}^{(k)}$ represents the coordinates of the alpha carbon for the residue with position number k found within protein *Q*. The value of n is the number of residues in the protein chain.

Animation List

The script that generates the animation will selectively display each of the models, one at a time, each model being displayed for a fraction of a second (try 0.2 seconds). The animation list will specify the order in which the models are to be displayed. We use the dissimilarity array to generate a list with consecutive models having the least RMSD. If the list is to work with *all* the models each appearing once and only once then the requirement that consecutive models have the least RMSD essentially makes the generation of the optimal ordering very difficult (think about the Traveling Salesperson Problem). So, we adopt the following heuristic:

- H 1): Start the list with two entries being the indexes designating the two models that are closest to each other (have the smallest dissimilarity).
- H 2) Find a model that is not yet in the list and that is closest to a model that is designated by either the entry at the start of the list or by the entry at the end of the list. If this found model is closest to the model designated by the entry at the end of list, then append its index to the end of the list. If this found model is closest to the model designated by the entry at the start of the list, then insert its index at the beginning of the list.

Repeat step (H 2) until all model indexes are put into the list.

Doing the Animation

Use `runCommand` with the string: `"modeldisp #" + idStr + "." + subIDstr` to selectively display a model. The variable `idStr` designates its id and `subIDstr` designates its subid. For example, in 1BMR the model with index 0 has id 0 and subid 1 so the `modeldisp` command has argument "#0.1" as shown in the above string. We can make this model disappear by using the string: `"~modeldisp #" + idStr + "." + subIDstr` as an argument for `runCommand`.

To generate the pause that should occur before the removal of a model you can use the `time.sleep(0.2)` statement. To use this, you will have to have

```
import time
```

at the start of your script. You can avoid the large transition that may be seen in going from the model at the end of the list to the model at the start of the list by running the list in the forward direction and then doing the animation again but this time going through the list in the reverse direction. The `numCycles` argument of `doAnimation(ani_List, numCycles)` represents the number of times the animation will cycle up and down the animation list.

As usual, hand in the script to the TA so that she can run your program to verify that it works. You can test your script on any or all of the following: 1BMR, 1DW5, 1QDP, 1D1H, 1N4N, 1IJC, 1PFD, 2LI3, 1AH1.

End of Assignment 3.