Assignment 1 – Biomolecular Structure and Visualization

BIOE/BIOMEDIN/BIOPHYS/CME/CS 279

Due: **Thursday, October 17, 2024 at 1:00 PM**

The goal of this assignment is to familiarize yourself with the visualization software PyMOL and the basics of biomolecular structure and dynamics.

1 Preliminaries

- Exercise 1 involves programming and does not require a written response. **You will be asked to submit the code you wrote for Exercise 1 at the end of the assignment**. More details are under Section 9 (Submission Instructions), so please read this section carefully.
- See the Getting Set Up document for information on using all our course platforms (Ed Discussion, Gradescope, QueueStatus, Canvas). You will submit this assignment via Gradescope. Please contact the TAs via a private Ed post if you do not have access to Gradescope.
- **Computing options:** We have found that software installation for these assignments can be challenging for students. Even if you don't have software installation challenges on this assignment, you may have them on assignments 2 and 3. As a result, we **strongly recommend** using the Stanford LTS computers, which have all the necessary software pre-installed. Our suggestions for computing options in order of preference are:

    1. Using in-person LTS machines: We strongly recommend using these machines for this assignment if you live on-campus or can physically access on-campus buildings. Here are links to the full list of LTS locations and the library hours for using the in-person LTS machines. The LTS machines in residences are available 24/7, as are the machines in the Lathrop study room. Once logged into the LTS machine, download the assignment files from the course website. **Make sure to save your files to Google Drive or an external drive. Files downloaded onto the LTS machines are deleted upon logging out.** The LTS machines use PyMOL2.5.
    2. Self-installed software: If you are not able to use the in-person machines, have a Mac or Linux machine, and are comfortable installing software using the command line, you can try self-installing software. This requires installing PyMOL2.5 and Anaconda for Python3.9 as outlined in the software handout. **Make sure you are using PyMOL2.5, as other versions of PyMOL are not compatible with the assignment!** Carefully follow the instructions to download the correct version. Software installation will be

harder for assignments 2 and 3, so we suggest getting familiar with LTS machines now if at all possible.

3. Remote LTS access: The number of LTS machines available for remote access is very limited. As a result, there could be a shortage if too many students use the LTS remote machines shortly before the due date, so please plan to complete the assignment early. See the instructions in working on LTS computers for setup details. Once logged into the remote LTS machine, download the assignment files from the course website. **Make sure to save your files to Google Drive or an external drive. Files downloaded onto the LTS machines are deleted upon logging out.**

- Download the assignment zip file from the course website.(`http://web.stanford.edu/class/cs279/index.html#hw`)

- **Start early to check your software setup**. Please start the assignment early so that you can take advantage of the available course resources in case you run into technical difficulties. **In particular, make sure you can run PyMOL2.5**. If you are working locally, you will also need to install matplotlib inside PyMOL, which this assignment will walk you through before Exercise 1. If you run into PyMOL installation issues, first double check that you correctly followed the installation instructions on the software handout. If you encounter further issues, please make a post on Ed or come to office hours.

  Please test your chosen computing and software setup by Monday 10/7 so that you can ask the TAs for help by then if necessary.

2 Visualizing Proteins

To start, we will use the program PyMOL to visualize the molecular structures of some polypeptides and proteins. PyMOL allows you to execute instructions through its GUI or through its built-in command line.

Please refer to the the PyMOL Wiki for detailed PyMOL documentation and PyMOL Reference Card as a cheat sheet for PyMOL commands and descriptions.

First, open PyMOL and also unzip the contents of the assignment. In the PyMOL command line, navigate to the "pdbs" folder in your assignment directory using **cd**. See the software handout for instructions on using **cd** if you are new to terminal commands. Then use the **load** command to import the HRas protein into PyMOL.

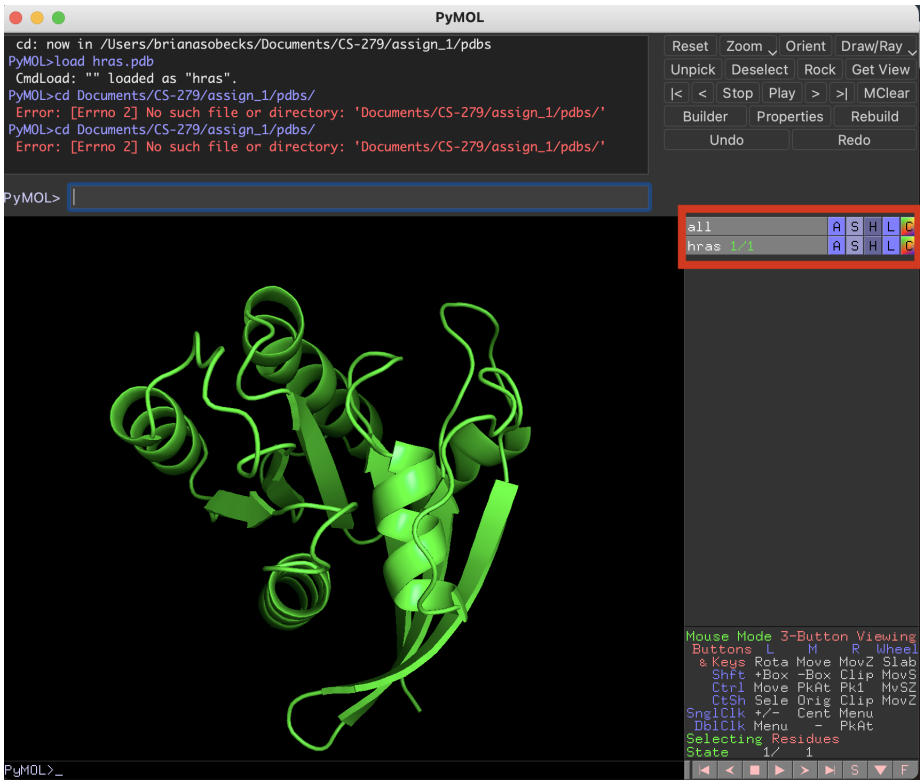
```
PyMOL> cd <path to assignment directory>/pdbs
PyMOL> load hras.pdb
```

When you load a pdb file, the default visualization is a **cartoon** depiction of the structure. You can toggle between different depictions of the same structure by hiding the current depiction and then showing a different depiction.

cartoon view provides a visualization of secondary structures present in the folded 3D conformation.

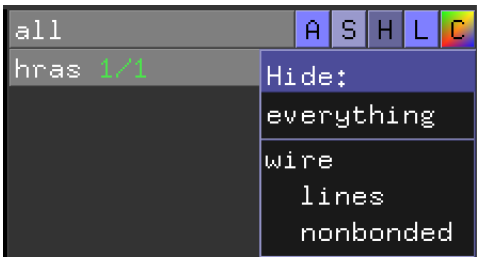
wire or **licorice** view provides a detailed view of atoms and bonds within the structure.

The commands to toggle between different views are found under the **show (S)** and **hide (H)** tabs on the right side of the GUI. See the red box in the image below for the location.



If we want to toggle to the wire view for instance, we would find the row of the selection we wish to manipulate (in this case `hras`).

If you click one of the buttons (in this case the H button), you will see a popup menu like this. (Note that the image shown here is cropped to take less space. You will have more options on your own screen.) Then select the following commands.



`H` → `everything` `S` → `wire`

If you prefer, you can also control the depiction from the command line.

```
PyMOL> hide everything, hras
PyMOL> show licorice, hras
```

Next, add hydrogen atoms to our view. Hydrogen atoms are absent in most pdb files, as most experimental structure determination techniques cannot resolve hydrogen atoms. We will use the command **h_add** to add hydrogen atoms. **h_add** uses a rudimentary algorithm to infer the positions of hydrogen atoms. Once added, hydrogen atoms should be represented in white.

```
PyMOL> h_add
```

Toggle between the cartoon view and the wire view. You should notice that the hydrogen atoms are visible in the wired representation but not in the cartoon representation.

```
PyMOL> hide everything, hras
PyMOL> show cartoon, hras
```

Switch to the cartoon view for now. In cartoon view, you should see alpha helices depicted by twisting coils and beta strands (which make up beta sheets) depicted with thick arrows.

Question 1:

(a) How many separate alpha helices does HRas have?

(b) How many separate beta strands does HRas have?

Note: You may find it useful to color by secondary structure.

`C` → `by ss` → `Helix Sheet Loop`

Alternatively, you can use the command in PyMOL:

```
PyMOL> color red, ss h; color yellow, ss s
```

Note: *'ss' specifies secondary structure selection, 'h' corresponds to alpha helices, and 's' corresponds to beta sheets.*

3 Secondary Structure Elements

Next, we will take a closer look at the most common secondary structures, the alpha helix and the beta sheet. Start by reinitializing PyMOL and loading helix.pdb.

```
PyMOL> reinit
PyMOL> load helix.pdb
```

Question 2: *Orient the structure so that you are looking through the helix (imagine looking through a telescope). Which direction (clockwise or counterclockwise) is the helix coiled as it extends away from you? Hint: Click and drag to change the orientation of the molecule.*

The orientation of the backbone atoms (ignoring side chains) tells us a lot about the overall structure of a protein. To only select the backbone atoms, execute the following commands.

```
PyMOL> hide everything
PyMOL> select bb, name c+n+ca
PyMOL> show licorice, bb
```

Note: The PyMOL selection syntax is:

```
PyMOL> select <name given to selection>, name <selection>
```

We are selecting all the atoms c, n, and ca in the structure and naming this selection `bb` which stands for backbone. Your **selection** can be defined by the atom name (**name n** to select all nitrogen atoms and **name n+ca** to select all the nitrogen and $C\alpha$ atoms). We can then manipulate/view/hide a selection as we would the full structure. The **lines** view shows each atomic bond as a single colored line.

The backbone geometry can be succinctly described by the *dihedral angles* (also known as torsional angles), which are labeled as ϕ, ψ, and ω (Figure 1). In particular, suppose you have four atoms (X, Y, Z, and W) with X bonded to Y, Y bonded to Z, and Z bonded to W. If you're "looking down" the bond between Y and Z (so that Y and Z appear to be on top of one another), the dihedral angle is the angle formed by the other two bonds (i.e., the X-Y bond and the Z-W bond).

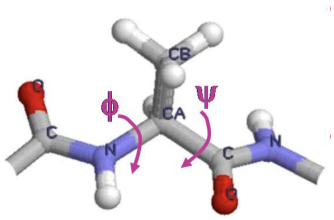


Figure 1: A diagram showing the ϕ and ψ dihedral angles.

Dihedral angles can be calculated in PyMOL using the **get_dihedral** command. To perform this calculation, first type **get_dihedral** in the PyMOL command line. Next, use the **pkAt** mouse command (by double-clicking with the right click) to select the four atoms, in correct order, involved in calculating the dihedral angle.

If you are working on a laptop, you may find it easier right click on `Mouse Mode` in the bottom right corner and set

`Mouse Mode` → `2 button viewing mode`

in which case **pkAt** can be performed with a single click and resetting the atom selection can be done with shift+click. In any mouse mode, you can view the available mouse commands in the

bottom right corner of the PyMOL window.

Notice that when you click an atom, PyMOL prints out the name of the atom you clicked, including the residue name, residue number, and atom name. The precise format is as follows:
`/system name/segment ID/chain name/residue name‘number/atom name`

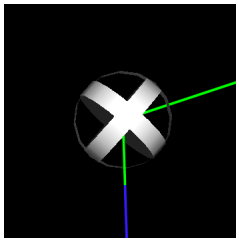


Figure 2: An atom selected using the **pkAt** mouse command.

After selecting your four atoms, the angle is shown graphically, and you can calculate a more precise angle by using the **get_dihedral** command.

```
PyMOL> get_dihedral (pk1),(pk2),(pk3),(pk4)
```

Alternatively, PyMOL has a powerful atom selection language. You can select by residue number using

```
PyMOL> select resid 159
```

and by atom name using

```
PyMOL> select name ca
```

Combining these into one command, select the $C\alpha$ of residue 159 using

```
PyMOL> select resid 159 and name ca
```

or the equivalent short form

```
PyMOL> select 159/ca
```

The $C\alpha$ atom of an amino acid residue is the backbone carbon atom to which the side chain connects.

You can then combine these selections to compute a dihedral with **get_dihedral**.

```
PyMOL> get_dihedral resid1/atom1, resid2/atom2, resid3/atom3, resid4/atom4
```

Question 3: *Calculate the dihedral angles for residue 159 in helix.pdb. Round to 3 decimal places.*

(a) Which amino acid type does this residue correspond to?

(b) Compute the dihedral angle ϕ_{159} *(between* $C_{i-1} - N_i - C\alpha_i - C_i$*).*

(c) Compute the dihedral angle ψ_{159} *(between* $N_i - C\alpha_i - C_i - N_{i+1}$*).*

(d) Compute the dihedral angle ω_{159} *(between* $C\alpha_i - C_i - N_{i+1} - C\alpha_{i+1}$*).*

Next, we will perform the same computations for the beta sheet segment. Load the beta sheet from `pdbs/bstrand.pdb`. You should see two beta stands.

Question 4: *Calculate the dihedral angles for residue 41 in bstrand.pdb, similarly to how you calculated angles in helix.pdb.*

(a) Compute ϕ_{41}*.*

(b) Compute ψ_{41}*.*

(c) Compute ω_{41}*.*

4 Ramachandran Plots, and Coding in PyMOL

Now that we have computed the dihedral angles for two amino acid residues, it is natural to ask whether these measurements are typical. In fact, one way of characterizing proteins' secondary structures is by their ϕ (phi) and ψ (psi) angles.

A common way to represent the distribution of backbone dihedral angles is a **Ramachandran plot** (also known as a Ramachandran diagram). Ramachandran plots are two-dimensional heat maps, which represent the number of residues in a given set of structures for a given ϕ and ψ.

We will be implementing a new PyMOL command that computes a Ramachandran plot for a given selection. Before moving on, make sure you have installed the python package matplotlib if you are working locally. This can be done by running

```
PyMOL> pip install matplotlib
```

In the PyMOL python API, commands will generally follow the same syntax as in the normal PyMOL window, but will be preceded with "cmd." For instance, we can translate the dihedral command we used above into the following python code.

```
cmd.dihedral(sel1, sel2, sel3, sel4)
```

New commands can be added to PyMOL by implementing them in python in a script. For this section, the script is called "analysis.py", and is located in assignment directory. Notice the line at the top which says:

```
from pymol import cmd
```

This imports the command (**cmd**) module from PyMOL. There is another line at the bottom which says:

```
cmd.extend(ramachandran, 'ramachandran')
```

This adds your **ramachandran** function to the PyMOL command window, where you can call the function as needed. Running the below command will make the **ramachandran** command available in the PyMOL window. *This will need to be run each time after editing analysis.py in order for changes to take effect!*

```
PyMOL> cd ..  # to get back to assignment directory
PyMOL> run analysis.py
```

The Ramachandran function itself is in the middle of "analysis.py" and looks like this:

```
def ramachandran(selection):
    pass #code goes here!!
```

You will fill in this section with your implementation of the **ramachandran** function. More information on how to properly implement the function is written in commented code blocks within "analysis.py."

Exercise 1: *Implement the **ramachandran** function in analysis.py. You can test your implementation by applying it to the two cases you computed manually before by running*

```
PyMOL> ramachandran helix and resid 158-160
```

and then

```
PyMOL> ramachandran bstrand and resid 40-42
```

Running these commands should open a plot window with a single dot at the angles you found for the earlier questions. ***Note****: Make sure* `helix.pdb` *and* `bstrand.pdb` *are already loaded before running* `ramachandran`*.*

Now we will use our new **ramachandran** PyMOL command to generate Ramachandran plots for alpha helices, beta strands, and the full HRas protein. First, load the helices (`helices.pdb`), bstrands (`bstrands.pdb`), and full HRas structures from "pdbs/". Then, produce a Ramachandran plot for each.

Question 5: *Include all three Ramachandran plots in your submission, and circle and label or describe the regions in the plot for HRas which correspond to alpha helices and beta strands.*

Question 6: *Why are ω angles not plotted in Ramachandran plots?*

Hint: How does the range of typical values for ω angles differ from range of typical values for ϕ and ψ angles? If you are unsure, choose a few residues from `hras.pdb` *and calculate the angles! See question 3 for the definition of an ω angle.*

5 Visualizing Nucleic Acids

While the previous sections have focused on protein visualizations, PyMOL also enables you to visualize many other biomolecules, including nucleic acids (DNA and RNA). In this section, we will visualize the **Cas9 guide RNA-DNA complex**.

In 2020, Emmanuelle Charpentier and Jennifer A. Doudna were awarded a Nobel Prize for their work on CRISPR-Cas9. Cas9 is the most commonly used enzyme for CRISPR gene editing, and its function is to cleave DNA at a targeted location. Cas9 functions in concert with a guide RNA molecule, which includes (among other elements) a short RNA sequence that binds to the target DNA sequence through complementary base pairing — A with U or T, and G with C. This use of a guide RNA is the main advantage of the CRISPR-Cas9 system compared to previous gene editing technologies, as it is much easier to design and synthesize a guide RNA that targets a particular DNA sequence than to modify a DNA-binding protein to target a new DNA sequence. For example, this enables researchers to target thousands of target sites in a single high-throughput experiment by using well-established technologies for nucleic acid synthesis and replication.

Let's examine the structure of the Cas9 guide RNA-DNA complex. The **fetch** command will download the structure directly from the PDB. This requires an internet connection.

```
PyMOL> reinit
PyMOL> fetch 5f9r
```

You should now be able to see cartoon representations of nucleic acids and the Cas9 protein. Similar to the atom selection language, PyMOL has a set of commands for interacting molecules. You can select molecules using

```
PyMOL> sele resname <molecule name>
```

or

```
PyMOL> sele resn <molecule name>
```

for short.

Since it is difficult to distinguish DNA from RNA at first glance, we will generate selections of DNA and RNA

```
PyMOL> select prot, polymer.protein
PyMOL> select rna, bymolecule resn U
PyMOL> select dna, polymer.nucleic & !rna
```

Question 7: *Hide the protein and color the RNA and DNA contrasting colors using the selections you just defined and PyMOL commands covered earlier in the assignment. Include a screenshot with just the RNA and DNA molecules visible in cartoon view, coloring the RNA a distinct color from the DNA (e.g. red and blue).*

Hint: use `color <color>, <selection>` *to set colors from the PyMOL command line.*

Notice that the guide RNA has separated the DNA double helix, such that the RNA bases pair with one strand of the DNA, while the other strand of the DNA is unpaired. This type of structure is called an R-loop, as shown in the figure below. This pairing of the RNA strand with the DNA strand is core to the mechanism by which Cas9 determines which DNA sequence to target.

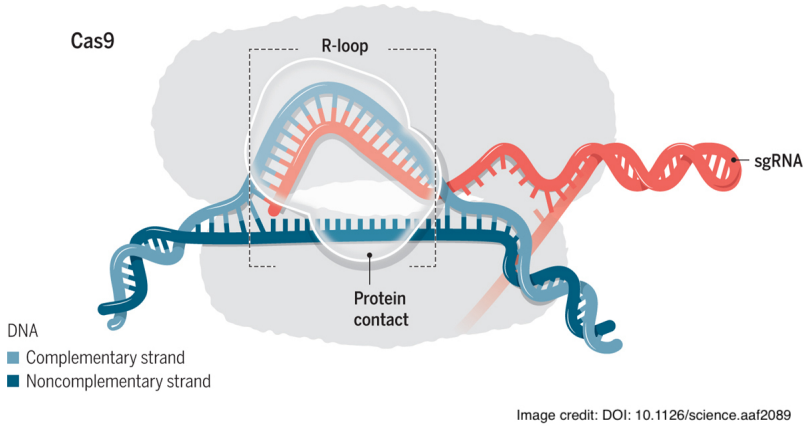


Figure 3: Diagram of R-loop formation during Cas9 binding

Question 8: *What is the name of the first unpaired DNA nucleotide after the RNA-DNA junction (end of the R-loop)? Give the name of the nucleotide on the unpaired strand, rather than the nucleotide paired to the guide RNA. Your answer should be A, T, G, or C.*

Hint: If you click on a DNA nucleotide in PyMOL, it will print the nucleotide's ID, including DA, DT, DG, or DC for DNA nucleotides A, T, G, and C respectively.

6 Structure and dynamics of the β_2 adrenergic receptor

The β_2 adrenergic receptor (B2AR) is an adrenaline detector. When adrenaline is present in the bloodstream, B2AR initiates signaling pathways that lead to relaxation of the airways and thereby increased breath volume. Drugs that mimic the effect of adrenaline on the B2AR, such as salbutamol, similarly stimulate increased respiration and are the active ingredient in asthma inhalers.

6.1 B2AR's home in the cell membrane

Adrenaline circulates outside of the cell membrane, but in order for cell behavior to be altered, changes need to occur inside the cell. B2AR is embedded in the cell membrane with a section protruding from the extracellular end that recognizes adrenaline (and other similar molecules) and a section on the intracellular end that recognizes a signaling protein called a G protein. When adrenaline binds to the extracellular section, it increases the likelihood of a G protein binding to the intracellular section. Binding of a G protein stimulates G-protein signaling, which initiates a signaling cascade leading to the physiological response.

We will now visualize a model of the B2AR in its membrane environment. This model was created as a starting point for molecular dynamics simulations of the B2AR (which we will view soon). Tip: We will use the structures here in multiple questions. We recommend not running the "reinit" command in between questions, or else you will have to reload all the structures again.

```
PyMOL> load pdbs/b2ar_full.pdb
```

To more easily visualize the different components of the system, let's color the carbon atoms of each chain a different color.

```
PyMOL> util.cbc element C
```

The conformation of the protein and ligand is based on an experimentally determined structure, but the other molecules were modeled computationally. Instead of adrenaline, a different ligand, named BI-16710, is present (`sele resname P0G`) in the adrenaline binding site. BI-16710 has the same effect on B2AR as adrenaline. (A "ligand" is any molecule that binds to a protein or other macromolecule. BI-16710 is a small molecule, as is adrenaline.)

Bound to the intracellular part of B2AR is a "nanobody". The nanobody has chain name "B". The nanobody is essentially a miniature antibody (a small protein) designed to interact with the receptor in a similar way that a G protein would. This nanobody facilitates experimental structure determination because it is smaller and more rigid than a G protein.

Question 9: *There are seven types of molecules present in this system. The seven types are B2AR, the nanobody, BI-16710, water, lipid (long fatty acid chains), sodium ions, and chloride ions. For each type of molecule, include an image of the molecule in the system and give the name of the molecule as shown in PyMOL. (For B2AR and the nanobody, give the chain name. For other molecules, give the residue name.) You can find the name by clicking on the component and reading the output in the terminal window. The output structure is as follows:*
`/system name/segment ID/chain name/residue name‘number/atom name`

The positions of hydrophobic and hydrophilic amino acid residues on the surface of the protein are essential for maintaining B2AR's position in the membrane. Visualize the positions of positively charged residues, arginine and lysine (`sele resname ARG+LYS`) (which are highly polar), and a few representative non-polar residues, valine, isoleucine, and leucine (`sele resname VAL+ILE+LEU`), by showing their stick representations and assigning them different colors. Then run the following commands:

```
PyMOL> hide everything, chain I
PyMOL> hide everything, solvent beyond 5 of polymer
PyMOL> hide everything, chain L beyond 5 of polymer
```

Now you can see the protein, lipids, and water much more clearly, which will help you for the next question.

Question 10: *How does the spatial distribution of positively-charged and non-polar residues differ, and how does this difference serve to stabilize B2AR in the membrane? In other words, where are the non-polar or positively-charged residues located relative to both the lipid and the water molecules? Phrase your answer in terms of the types of residues that interact with water or lipid molecules in the modeled orientation, and how this would change if the B2AR were turned on its side or removed from the membrane. Hint: The cell membrane is made up of two layers of lipid molecule. Each lipid molecule has a long tail that points into the membrane. These tails are highly hydrophobic. Notice where the water is located relative to the membrane.*

6.2 Structural changes associated with B2AR activation

Now we will consider two experimentally determined structures of the B2AR. One structure, which we will refer to as an "active-state" structure, was determined in the presence of an agonist—a ligand like adrenaline that stimulates G-protein signaling—and a nanobody (which mimics a G protein). The second structure, which we will refer to as an "inactive-state" structure, was determined in the presence of an antagonist—a ligand that binds to the receptor, but does not stimulate G-protein signaling—and no nanobody.

First, we can **fetch** the structures from the Protein Data Bank. (This step requires an internet connection.)

```
PyMOL> fetch 2rh1, inactive
PyMOL> fetch 3p0g, active
```

In addition to the protein and ligands, the inactive-state structure contains some molecules that are only present to facilitate experimental structure determination. Remove them using the below commands. (Note that the first two resnames being removed are "SO4" and "HOH" with capital O, not the number zero.) We will also add hydrogen atoms.

```
PyMOL> remove inactive and resid 1000-
PyMOL> remove resname SO4+HOH+GLC+12P+BU1+CLR+PLM+ACM
PyMOL> h_add active
PyMOL> h_add inactive
```

The relative orientation of the structures is currently random, making it hard to compare the structures. To make comparison easier, we will now align the inactive-state structure to the active-state structure. The alignment superimposes the structures as closely as possible.

```
PyMOL> align inactive, active
```

Let's look closely at the intracellular part of the protein where the nanobody is bound in the active-state structure (and where a G protein would bind). Use the following commands to make a clear visualization:

```
PyMOL> select nanobody, active and chain B
PyMOL> color [fun-color-name], nanobody
```

An object called `nanobody` will appear in the panel on the right. In the PyMOL GUI, select the following commands:

```
nanobody -> Actions (the button labeled A) -> extract object.
```

You will now see an object labeled `obj01`. Try clicking on either `active` or `inactive` to toggle the receptor showing/not showing. Notice how the nanobody stays on the screen even when the receptor is hidden. Finally, run

```
PyMOL> show surface, active and resid 266-281
PyMOL> show surface, inactive and resid 266-281
```

You may also want to recolor the residues in the selection above for easier viewing. Look at where

the surface is located relative to the nanobody. You can show the nanobody as a surface too if it's easier for you to visualize it that way. Then answer the following question.

Question 11: *The largest difference between the active and inactive states of the receptor involves an outwards (with respect to the center of the protein) movement of residues 266 to 281. How does this change allow the nanobody to bind? A brief, simple explanation is sufficient.*

Now let's look at the ligand binding pocket, where adrenaline and related ligands bind. Note that the agonist (ligand in the active-state structure) has resname "P0G", and the antagonist (ligand in the inactive-state structure) has resname "CAU".

Question 12: *The largest difference in the ligand binding site involves residue 207 (a serine). How does the position of residue 207 relative to the ligands change between the inactive- and active-state structures? What type of inter-molecular interaction does the agonist form with S207 that the antagonist cannot due to differences in their chemical structures? Show the ligands and S207 as sticks and look at which atom types are closest between the two. Hint: To see or change the coloration of atoms, select* `C` → `Color:` → `by element` *in the GUI. In stick representation, carbon should be whatever the base color of the object is, oxygen is red, nitrogen is blue, and hydrogen is white.*

6.3 Dynamics of B2AR

Now we will analyze a pair of molecular dynamics simulations of B2AR. Both the simulations were started from the active-state structure, but in one simulation the nanobody is retained and in the other the nanobody was deleted prior to beginning the simulation.

Waters, lipids, sodium and chloride ions, and hydrogen atoms were present when computing the simulation trajectories, as in the complete system we viewed earlier, but they are not included here to decrease the required memory.

These simulations were used in a paper entitled "Activation mechanism of the β_2-adrenergic receptor"; if you're curious, you can find it at https://www.pnas.org/doi/full/10.1073/pnas.1110499108.

6.3.1 Visualizing the simulations

Load the nanobody-bound simulation and align it to the active-state structure. The `intra_fit` command aligns all frames of a simulation to the first frame. The frames are snapshots of atomic coordinates saved at different points in time during the simulation. Note: We use the word "frame" to refer to a snapshot of atomic coordinates at one point in time during a simulation; in PyMOL, frames are referred to as "states".

```
PyMOL> load pdbs/b2ar_nanobody_bound.pdb
PyMOL> align b2ar_nanobody_bound, active, mobile_state=1
PyMOL> intra_fit b2ar_nanobody_bound and chain A and name CA
```

Then, do the same for the nanobody-free simulation.

```
PyMOL> load pdbs/b2ar_nanobody_free.pdb
PyMOL> align b2ar_nanobody_free, active, mobile_state=1
PyMOL> intra_fit b2ar_nanobody_free and chain A and name CA
```

Visualize the simulations! You can watch the simulation proceed either by pressing the play button at the bottom right of the window or by using the slider at the bottom of the window.

Each pair of consecutive frames are separated in time by 20 ns. A time step of 2 fs was used in these simulations, so there are 10 million simulation time steps between consecutive frames.

Question 13: *In one of the simulations, the receptor remains close to the active-state structure, whereas, in the other simulation, the receptor transitions towards the inactive-state structure. In which simulation do you observe the transition towards the inactive state? Focus on the large rearrangement of residues 266 to 281 that we looked at earlier. It will be useful to view the simulations with the active- and inactive-state structures (or at least one of the two) included in the visualization.*

6.3.2 Analysis of RMSDs

As you can see, in an MD simulation—as in real life—the atoms are in constant motion! The most popular measure of the overall similarity/difference between two structures is the root-mean-square deviation (RMSD).

We've provided a command for plotting RMSDs in "analysis.py". Remember, to use the script, you must first run

```
PyMOL> run analysis.py
```

to add the new commands. You can then produce an RMSD plot by running

```
PyMOL> plot_rmsd <reference selection> <trajectory name>
```

where the `<reference selection>` would refer to the structure that you want to use as a reference and `<trajectory name>` would refer to the simulation trajectory. For instance, to compute the RMSDs of the trajectory to the active-state structure for $C\alpha$ atoms, you would run

```
PyMOL> plot_rmsd active and chain A and name CA, b2ar_nanobody_bound
```

Note that in the structures and simulation frames, the receptor is assigned chain A and the nanobody is assigned chain B, so if we want to compute the RMSD for the receptor, we must specify chain A.

Question 14:

(a) Plot the RMSD of the $C\alpha$ atoms between the active-state structure and each frame of the nanobody-bound simulation. Do the same for the RMSD to the inactive-state structure. Include both plots in your write-up.

(b) Perform the same analysis for the nanobody-free simulation. Include both plots in your write-up.

6.3.3 Monitoring distances

Finally, it is often useful to monitor the behavior of particular parts of the protein of interest. A widely used metric for quantifying motion of particular parts of a protein is simply the distance between two atoms. Here, we will look at a distance that monitors the movement of residue 271, which as we saw earlier is positioned quite differently in the active- and inactive-state structures. We've provided a function in "analysis.py" to plot distances over a simulation. The function can be used, for example, using the below command

```
PyMOL> plot_distance b2ar_nanobody_bound, 131/CA, 271/CA
```

Question 15:

(a) Plot the distances between the Cα atoms of residue 131 and residue 271 for both the nanobody-bound and nanobody-free simulation. Include the plots in your write-up.

(b) Again, this result should agree with your previous visual assessment. What information do these distance plots capture that the RMSD plots do not?

Question 16: *The paper that includes these simulations observed similar behavior in repeated nanobody-free simulations. Also, even when these simulations were run much longer, no transitions were observed in the reverse direction (that is, from the inactive state to the active state). If you assume these simulations are highly accurate, what can you conclude about the relative free energies of the active and inactive states when the agonist is present but the nanobody is not? In particular, can you conclude that one of them is larger than the other (and if so, which)?*

7 Feedback

Please fill out the assignment feedback form and provide the code word below. We encourage constructive criticism.

Question 17: *What is the feedback form code word?*

8 Challenge Question

This question is intentionally more challenging than the rest of the assignment. It is not required, and you can receive full credit for the assignment without attempting it. We will never deduct points for a challenge question submission and will award extra credit depending on the quality of your solution.

Question 18: *Write a Python script that computes the average solvent accessible surface area (SASA) for each amino acid residue type in PDB files. Create a bar chart graphing average SASA for each residue type in*

```
sasa_pdbs/*
```

The starter code can be found in the `calculate_average_sasa` *function in "challenge_problem.py".*

What trends do you notice for charged vs. noncharged residues? Notice that the size of a residue will also affect its SASA. Come up with a strategy to account for different residue sizes and plot an adjusted average SASA per residue as before. What differences in SASA do you notice after factoring in a residue's size?

Look at the PyMOL documentation for tools to calculate SASA. The following documentation on PyMOL's API may be useful: `https://pymolwiki.org/index.php/Get_Area`*. Additionally, don't forget to add hydrogen atoms to your molecules using* `h_add` *prior to calculating SASA.*

9 Submission Instructions

Please submit this assignment on Gradescope (https://www.gradescope.com/). Type up your answers in the text editor of your choice (LaTeX, Word, etc.), making sure to clearly note the question number associated with your answer, then convert the entire document to a pdf.

Go to Gradescope, navigate to CS279, "Assignments", then select **Assignment 1**. Choose the option to upload a pdf and **tag each question with the corresponding page**. You do not need to separate each question on its own page.

You should submit your code for Exercise 1 as well. Please append your code for the `ramachandran` function in text format to the end of the assignment write-up, and tag the corresponding pages under "Exercise 1". No other code is required to be submitted for this assignment.

If you choose to tackle the challenge question, please include your results and written explanation for the challenge question in your main pdf. Please submit the "challenge_problem.py" file for the challenge question on Gradescope separately, under **Assignment 1 Challenge Question Code.** Let the course staff know if you have any issues.