Making Ramachandran Plots

Bootcamp Summer Programming Project

Introduction

The purpose of this project is to give you some practice using Python in a scientific context. We designed the project for beginning programmers, but before starting you'll still have to teach yourself the basics of Python.

The project is to write a program that generates Ramachandran plots from Protein Data Bank (PDB) files. Ramachandran plots show how backbone torsion angles are distributed in proteins. Different features of proteins (namely α-helices and β-sheets) occupy distinct regions in these plots. For this reason, Ramachandran plots are commonly used both to validate experimental structures and to predict computational structures.

The project is divided into four parts:

1. Setting up Python and reading the command line.
2. Parsing groups of four backbone atoms from PDB files.
3. Calculating torsions angles from those groups of atoms.
4. Plotting those torsion angles using matplotlib.

Try to get through at least one part every two weeks. It takes time to get comfortable writing code, and you won't get there if you put off the whole project until just before bootcamp starts. The project will be due on the first day of bootcamp. We will wrap up the project with an activity where everyone explains their code to a partner (i.e. a code review). This activity won't be as valuable for you if you don't finish the project on time. If you are worried about finishing the project on time, or have any questions about the project or programming in general, please don't hesitate to email any one of us.

What if you already know how to program?

Even if you already know how to program, it's still important for you to work through this project so that you can participate in the code review we'll be holding on the first day of bootcamp (see above). In fact, you should make an extra effort to make your code as clean and understandable as possible so that it can be an example for the more novice programmers in your class.

If you know how to program, but not in Python, take this opportunity to learn Python. The challenges in the *Future Directions* section will be especially useful to you, because they will make you think a little bit more and really explore the language.

What is a Ramachandran Plot?

Let's begin with some basic terminology: proteins are chains of amino acid residues. Figure 1 shows three residues from a larger protein. A distinction is made between the backbone (blue) and the sidechains (red). Every residue in the protein is connected through the backbone, which has a very regular pattern of atoms: $\mathrm{N{-}C_\alpha{-}C{-}N{-}C_\alpha{-}C{-}\ldots}$ where the carbon that supports the sidechain is called C_α to distinguish it from the one that doesn't.

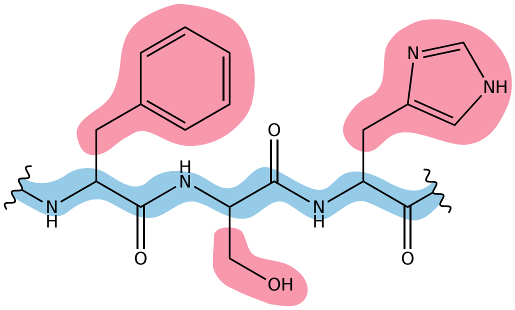

Figure 1: Three residues with the backbone and sidechains highlighted.

Torsion angles are a convenient way to describe the geometry of the backbone. A torsion angle can be intuitively described as the amount of twist around a bond. To be more precise, consider four consecutive backbone atoms. You always need four atoms to define a torsion angle. Use the first three atoms to construct one plane and the last three atoms to construct another. The angle between those two planes is the torsion angle, or the twist around the bond connecting the second and third atoms. These angles are a convenient way to describe geometry because they don't depend on the orientation of whole protein in Cartesian space.

For each residue in a protein, three backbone torsions are defined:

Torsion	Backbone Atoms
ω_i	C_α^{i-1}–C^{i-1}–N^i–C_α^i
ϕ_i	C^{i-1}–N^i–C_α^i–C^i
ψ_i	N^i–C_α^i–C^i–N^{i+1}

Table 1: Backbone torsion angles.

A Ramachandran plot is a scatter plot showing the ϕ and ψ angles for each residue. The ϕ angles are plotted along the x-axis and the ψ angles are plotted along the y-axis. Each point is the ϕ_i, ψ_i pair for one residue. The ω angles are left out because they never deviate much from 180°.

Figure 2 shows the Ramachandran plot for chain A of PDB ID: 1AXC, which is one of the structures we uploaded for you to use. Note that there are two distinct clusters of points. These represent two specific backbone conformations that are common in folded proteins: α-helices (lower cluster) and β-sheets (upper cluster). The ability to easily identify which residues are contributing to well-known, stable structural motifs is what makes Ramachandran plots useful for structural validation and prediction.

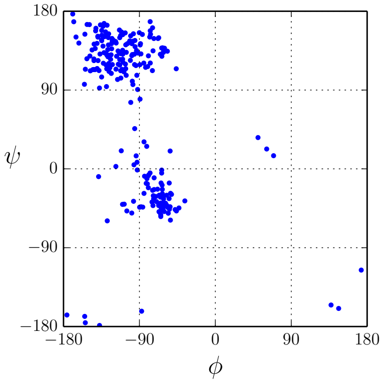

Figure 2: Ramachardran plot for chain A of PDB ID: 1AXC

What Do You Need to Do?

Your task is to create a program written in Python that takes a PDB file (specified on the command line) as input and produces a Ramachandran plot as output. Use the paragraphs below to guide how you write your program.

Part 1: Using Command Line Arguments

Before anything else, you need to install Python on your computer and learn how to run a “Hello world!” program. Exercise 1 of Rosalind explains how to do this pretty well for a variety of systems, but if you get stuck email one of us and we’ll make sure you can get started. Once you’ve done this, work through as much of Rosalind as you can. If you’re learning how to program for the first time, we recommend that you take notes as you work through the exercises.

Your program should accept the name of a PDB file as a command line argument. This will allow you to easily generate Ramachandran plots for different structures without having to change your script itself.

Before moving on...

1. Write a program that just prints the name of a file specified on the command line. For example:

   ```
   $ python rama.py 1AXC.pdb
   1AXC.pdb
   ```

   Here the $ sign represents the shell prompt. The rest of that first line is what you should type into the terminal to run your program. The second line is the output that your program should produce.

Part 2: Reading PDB Files

PDB files contain information about the 3D structure of proteins. PDB stands for Protein Data Bank, which is an online database of protein structures. These files are regular text files (so you can view them with the same program you use to edit Python code) where each line contains different information about the structure. The lines containing the information relevant to this project all begin with the keyword ATOM:

```
         1         2         3         4         5         6         7         8
12345678901234567890123456789012345678901234567890123456789012345678901234567890
ATOM   2305  CA  ILE A 255      35.578   9.357   5.792  1.00 43.22           C
```

Listing 1: An example ATOM record.

Listing 1 shows an example ATOM record. (The gray numbers aren’t part of the record; they’re just numbering the columns.) Each ATOM record identifies a particular atom and specifies its coordinates. Different pieces of information about the atom are given at different offsets from the beginning of the line. For example, the 13th - 16th characters of our example (`CA`) specify that the atom in question is a C_α. The 18th - 20th characters (`ILE`) specify that the atom is part of an isoleucine residue. Table 2 describes the `ATOM` record fields that are most relevant to this project. The complete PDB file format is also available online if you're interested.

Columns	Example	Description
1 - 6	`ATOM`	Record type
7 - 11	`7507`	Atom serial number
13 - 16	`CA`	Atom name
18 - 20	`ILE`	Residue name
23 - 26	`255`	Residue sequence number
31 - 38	`35.578`	X coordinate of atom
39 - 46	`9.357`	Y coordinate of atom
47 - 54	`5.792`	Z coordinate of atom

Table 2: Description of the fields in an `ATOM` record.

Learning how to break big problems into smaller problems is an important skill for new programmers to practice. This part of the project is a good place to do so because there are lots of ways to take small steps and get intermediate results when parsing a file. The milestones listed below break down some of your first steps, but they may still be challenging. If you get stuck, think about ways to break down the problem even further.

This is also a good time to focus on writing good comments. A good rule of thumb is that you should have a comment describing anything you needed to look up when writing the code in the first place, because when reading the code months from now, those are probably the things that will confuse you the most. Remember, we'll be going over this project on the first day of bootcamp. You'll get more out of that session if you can remember why you wrote your program the way you did.

Before moving on...

1. Print out all the lines in the PDB specified on the command line.
2. Extract the x, y, and z coordinates for each atom in the PDB file.
3. To calculate each ϕ and ψ torsion, you need coordinates for four atoms. Table 1 shows which coordinates are needed for which torsion. Make groups of all the coordinates you'll need to calculate each torsion angle.

Part 3: Calculating Torsion Angles

Once you've read in Cartesian coordinates for all the backbone atoms, you'll need to calculate torsion angles. You can find the algorithm for this calculation here. Notice that this algorithm involves a healthy dose of vector arithmetic. Since Python doesn't have built-in support for vector arithmetic, you'll have to write and test your own functions to perform the tasks outlined in Table 3. Listing 2 shows what one of these functions might look like. Keep in mind that a vector is nothing more than a set of three coordinates, like the tuples we used above to store the backbone atom coordinates.

Function	Example Input	Example Output
Add two vectors	`(1, 2, 3) (3, 4, 5)`	`(4, 6, 8)`
Subtract two vectors	`(1, 2, 3) (3, 4, 5)`	`(-2, -2, -2)`
Normalize a vector	`(1, 2, 3)`	`(0.27, 0.53, 0.80)`
Calculate a dot product	`(1, 2, 3) (3, 4, 5)`	`26`
Calculate a cross product	`(1, 2, 3) (3, 4, 5)`	`(-2, 4, -2)`

Table 3: Vector arithmetic functions.

```python
# v1 and v2 are both 3D vectors represented as tuples of floats
def add(v1, v2):
    return v1[0] + v2[0], v1[1] + v2[1], v1[2] + v2[2]
```

Listing 2: Example addition function.

Once you have written these functions, use them to create another function that calculates a torsion angle using the algorithm linked above. This function should take four vectors as input and return a single angle as output.

Before moving on...

1. Write and test the functions listed in Table 3.
2. Write and test a function that calculates the torsion angles using the algorithm linked above. This function should accept four tuples of three floats each and should return a single float.
3. Create two lists: one that contains all the ϕ torsions and one that contains all the ψ torsions. This should build on the two loops you wrote at the end of the previous section.

Part 4: Making Plots

Use `matplotlib` to generate your Ramachandran plot. Rosalind doesn't cover `matplotlib`, so you'll have to figure out how to install it and use it on your own. Here are the official installation instructions and here is a pretty good tutorial that shows how to do the kind of things you'll need for this project. Again, please don't hesitate to email one of us if you get stuck on any of this.

Future Directions

If you found this project to be useful and you want to continue improving your scientific programming skills, try one or more of these extra challenges:

1. Plot interesting subsets of residues

It can be informative to make Ramachandran plots for subsets of the residues in proteins. For example, glycines and residues that come just before prolines (i.e. pre-prolines) have very characteristic Ramachandran plots. Modify your program to either show just glycines, just pre-prolines, or everything except glycines and pre-prolines. There won't be enough residues in a single structure to really show what these plots should look like, so try combining this with the challenge below to fill in your plots. Also, if you don't already know, you might find it interesting to ponder why glycines and pre-prolines have distinctive plots.

2. Use NumPy to speed up the vector arithmetic

NumPy is a fast and comprehensive Python module for performing vector and matrix calculations. Not only can using NumPy greatly simplify your code, but it also can drastically speed it up. Also, most scientific computing and visualization packages in Python are NumPy-compatible out of the box. Try using NumPy arrays instead of tuples to store your vectors, and then use NumPy functions for the vector math. There are plenty of introductory NumPy tutorials on the web. The resulting speed-ups will help with the next future direction.

3. Make a combined plot for 10,000 structures

Working with lots of data is an important skill for scientific programmers. Toward that end, we uploaded a subset of the CATH database comprising 11,926

PDB files to the bootcamp website. Download this data and modify your program to create a single Ramachandran plot for all of it. Experiment with ways to quickly process so many files and to clearly present so many data points.

4. Automatically download PDB files

One of the nice things about Python is that it's not just for scientific programming, so there are libraries for doing things like downloading files from the web. The PDB in particular makes it very easy to automatically download structures. Modify your program to accept a PDB ID rather than an actual file, then download that structure from the PDB and use it to generate the plot. The `requests` library is a good place to start.