# Generation of 3D Structural Descriptors of Protein Conformations from NetCDF-formatted MD Trajectory Files

Abdul-Rashid B. Sampaco III

CHEM 359: Computer Programming for Various Applications in Chemistry
Programming Project / Final Exam

MS Chemistry
Institute of Chemistry
University of the Philippines

## I. INTRODUCTION

Since Anfinsen's seminal findings on the thermodynamic hypothesis of protein folding, the field of protein folding and structure prediction has been an area of active research. Scientists have tried to device a workflow for locating a protein's native state given just its amino acid sequence. Among the methods that have been explored so far include energy minimization and sampling methods like molecular dynamics (MD), Monte Carlo (MC), Simulated Annealing (SA), and Genetic Algorithm (GA). Energy minimization methods make use of the idea that the native state of the protein is the global energy minima in its energy landscape. Several groups have tried developing different energy functions and finding their minima. These energy functions usually contain terms involving bond lengths, bond angles, and interactions. However, proteins are large molecules and thus their thermodynamics are not described by potential energy or enthalpy alone. Furthermore, at nonzero temperatures, entropic contributions to a system's thermodynamics constantly increase. Therefore, the thermodynamics of protein folding is better defined by its free energy landscape. Calculation of free energy is a lot trickier than calculation of potential energy. Free energy is a property of the system rather than of a single conformation. To achieve accurate free energy calculations, the sampling method to be used, aside from being able to sample the entire conformational space, should also be able to retain the canonical probability distribution of the protein's states. This is very hard to achieve especially since proteins are large molecules that are very susceptible to kinetic traps.

An alternative approach to protein structure prediction may be to use quantities, aside from energy, that varies with the protein's conformation to score the structures according to how close they are to the native state. Some of the most obvious quantities that may change according to conformation include intramolecular noncovalent interactions. This may be a good start to exploring quantities that may potentially be used to score protein conformations. A large set of data (may come from any sampling method including MD, MC, etc) containing conformations of proteins with known experimental structures has to be collected. Different structural descriptors can then be correlated with some quantity describing how close they are to their native state (RMSD, etc). Ideally, this should lead to having a scoring function that can distinguish native conformations from others, and a workflow that can predict native structures straight from the amino acid sequence can finally be developed.

The first step towards this study is finding a computational tool that can accomplish what is needed, that is, to convert structural information contained in trajectory files into something quantitative. Today, the existing tools that can locate intramolecular interactions in proteins are 1) Protein Interactions Calculator (PIC) (Tina et al., 2007), a webserver that is available on http://pic.mbu.iisc.ernet.in/, and 2) ProtInter (Borry, 2017),

an open-source command line tool written in Python 3 that is available on https://github.com/maxibor/protinter.

PIC takes in PDB files as input and calculates interactions based on empirical and semi-empirical rules. Multimodel PDB files, as in PDB files of NMR structures, can be sent to the server as input, but it is only able to analyze the first structure. Thus, analysis of a large set of protein conformations entails manually uploading each single PDB file to the server. As such, given that it is a webserver, the performance of PIC does not scale well with the number of conformations to be analyzed. This precludes it from being used efficiently in protein folding studies where the data involved are on the range of thousands to millions of protein structures.

ProtInter is an open-source command line interface tool written in Python 3 that utilizes the library Biopython (Cock et al. 2009) to parse through PDB files. Being a CLI tool, it appears to be a much better option compared to PIC for protein folding studies as it allows for more customization. However, ProtInter suffers the same drawback as PIC in that it cannot handle multimodel PDB files. A workaround may be to run the program iteratively taking each structure as input at each iteration.

In my research, I use AmberTools for my MD simulations, and thus my data are formatted in NetCDF. NetCDF (http://www.unidata.ucar.edu/software/netcdf/) is a set of software libraries developed by the Unidata Program at the University Corporation for Atmospheric Research. It has APIs for several programming languages including Fortran, C, C++, and Python. If I wanted to analyze my data, calculation of intramolecular interactions using existing tools would require converting large NetCDF files into multiple single PDB files before they are fed into the tools. This is expected to be very inefficient as the existing data contains tens of millions of structures.

The best option is to develop a software that is specialized to handle large NetCDF files and can calculate interactions of each frame at a reasonable speed. This also allows for smooth transition of data from MD simulations straight to analyses. In this work, I develop a program that takes in two files as input: 1) the MD trajectory file, formatted in NetCDF as that used in AmberTools/AMBER, which contains the Cartesian coordinates of each frame, and 2) the parameter file, formatted in prmtop as that used in AmberTools/AMBER, which contains all parameters required to run an MD simulation. The program parses through both files to calculate structural descriptors that vary according to the protein's conformation. For each quantity, the program outputs two files. The first file contains the value of the quantity on each frame, while the second file contains a list of the atoms or groups involved in the interactions per frame. The program is written entirely in C++.

The structural descriptors that can be calculated by the program so far include:

**1. Hydrogen Bonds**

Hydrogen bonds are directional interactions between electronegative atoms covalently bonded to hydrogen atoms (hydrogen-bond donors) and other electronegative atoms. In typical protein structures, hydrogen bonds are formed from interactions of N-H groups with O atoms and O-H groups with N atoms. The presence of post-translational modifications involving other electronegative atoms like fluorine may introduce new forms of hydrogen bonds. This is not supported yet.

Hydrogen bonds were defined using a distance and an angle cutoff. N and O pairs should be less than 3.5 Å from each other, while the angle between them with the H as the vertex should be sufficiently obtuse at greater than 120°.

**2. Ionic Interactions**

Ionic interactions are long-range interactions of oppositely-charged particles at some distance from each other. Some amino acids with charged side chains include LYS, ARG, ASP, GLU, etc. Aside from these, the program also supports the less frequent, charged amino acid, protonated HIS (HIP).

The center of mass of the charged group of the amino acid side chains were taken to be the point representing the charged particle. Oppositely-charged particles less than 6 Å away from each other are considered to have an ionic interaction.

It should be noted that partial charges listed in whichever force field used in a simulation would deviate from their expected values. For example, the positively-charged side chain of LYS, which is expected to bear a +1 charge on its side chain N atom, would have a partial charge of less than +1 on the force field because of the presence of other groups in the molecule which may cause redistribution of electron density. The charge values on the force field are expected to be more accurate as these are based on empirical data. However, in this program, I do not consider these partial charges in defining ionic interactions and just stick with where the charges are expected to be.

**3. Aromatic-Aromatic Interactions**

Aromatic-Aromatic interactions are also known as the pi-pi interactions or pi stacking interactions. These involve interactions of phenyl rings among each other. There are three aromatic amino acids, PHE, TYR, and TRP.

An aromatic-aromatic interaction is defined as the event when the center of mass of the phenyl rings are between 4.5 and 7 Å from each other.

**4. Cation-Pi Interactions**

This interaction involves the interaction of positively-charged groups with the phenyl rings of aromatic residues. In this program, they are defined as the event when the cationic side chain of either LYS, ARG, or HIP is less than 6 Å away from the center of mass of a phenyl ring.

**5. Aromatic-sulfur interactions**

This interaction involves the interaction of the sulfur atom in CYS residues with the phenyl rings of aromatic residues. In this program, they are defined as the event when a CYS sulfur atom is less than 5.3 Å from the center of mass of a phenyl ring.

**6. Disulfide bonds**

Although disulfide bonds are covalent interactions as opposed to the other interactions considered in this program, location of the formation of possible disulfide bonds may prove to be quite useful in protein folding studies, especially in proteins with multiple expected disulfide bonds. If there is no available experimental structure for a protein, and thus the configuration of its disulfide bonds are not known, distance cutoffs can be used to determine whether in the course of a simulation, the CYS residues come close enough to each other for them to form a covalent bond. Obviously, this is still speculation and further reading is still required to realize its potential. Nevertheless, the program supports defining disulfide bonds based on distance cutoffs.

Disulfide bonds are defined as the event when two sulfur atoms of CYS residues are less than 2.2 Å away from each other.

**7. Hydrophobic Interactions**

Of all interactions supported in this program, this is the trickiest to implement. Unlike the other noncovalent interactions, hydrophobic interactions are not driven by attraction of chemical groups to each other, but rather they are a consequence of the favorable process of the dispersion of water molecules to the bulk liquid from having an ordered structure around nonpolar groups. Nonpolar chemical groups tend to clump together to have the least contact with water molecules.

In this program, hydrophobic interactions were defined as the event when C-H groups, that is, hydrocarbon groups, are within 5 Å of each other.

**8. Hydrophobic-Hydrophilic Ratio**

This is a quantity that is still on its experimental stages and is yet to be tested. It is still not known whether it has a significant physical representation or if it has been implemented properly as intended.

It was originally intended to describe a structure's compactness in terms of the hydrophobic interactions at its core and the hydrophilic interactions at its surface. The center of mass of the structure is first defined. Then, all C-H groups close to this center at less than 5 Å is considered to be hydrophobic interactions at the core of the protein. Furthermore, polar groups (N-H and O-H groups) that are away from this center at greated than 5 Å are also counted. The ratio of these two counts are taken and is reported to the output.

**II. DETAILS OF THE PROGRAM**

The program consists of two modules, **describe.cpp** and **vecop.cpp**, with two header files, **describe.hpp** and **vecop.hpp**, and the main program, **main.cpp**.

**A. describe.cpp, describe.hpp.** This module contains all classes, data types, and functions needed for the program to work.

Data Type: ***traj***.

This data type is where the information from both input files are stored. It contains:
1.) *coords*. three-dimensional vector for the coordinates (number of frames * number of atoms * xyz coordinates)
2.) *atoms*. string array of atom names
3.) *residues*. string array of residue names
4.) *charges*. vector of partial charges, and
5.) *masses*. vector of atom masses.

Class: ***Describe***.

This takes care of parsing of the input files and calculations of the desired quantities.

Functions:

1. *Constructor*. Takes the file names of the trajectory file and the prmtop file as inputs and directly parses both. It then saves the necessary information to pre-initialized containers.

The NetCDF parser makes use of an external module, the C++ API of NetCDF. The NcFile class was used to read the NetCDF files. The coordinates were extracted using the NcFile function getVar() and stored as a NcVar class. Further manipulations of the coordinate data were then done to store them to a 3D vector.

2. *HydrogenBonds (string, string)*. Takes into two strings, file names of the two output files, as input. Locates and counts the number of hydrogen bonds in a frame. Lists the atoms and residues involved in the hydrogen bonds found.

3. *IonicInteractions (string, string)*. Locates and counts the number of ionic interactions in a frame. Lists the atoms and residues involved in the ionic interactions found.

4. *AromaticAromatic (string, string)*. Locates and counts the number of pi-pi interactions in a frame. Lists the atoms and residues involved in the pi-pi interactions found.

5. *CationPi (string, string)*. Locates and counts the number of cation-pi interactions in a frame. Lists the atoms and residues involved in the cation-pi interactions found.

6. *AromaticSulphur (string, string)*. Locates and counts the number of aromatic-sulphur interactions in a frame. Lists the atoms and residues involved in the aroamtic-sulphur interactions found.

7. *Disulfide (string, string)*. Locates and conuts the number of possible disulfide bonds formed in a frame. Lists the atoms and residues involved in the disulfide bonds found.

8. *Hydrophobic (string, string)*. Locates and counts the number of possible hydrophobic interactions in a frame. Lists the atoms and residues involved in the hydrophobic interactions found.

9. *HydrophobicOverHydrophilic (string, string)*. Computes the ratio of hydrophobic groups near the center of mass and the hydrophilic groups near the surface. Lists the atoms and residues involved in the interactions found.

**B. vecop.cpp, vector.hpp.** This module contains functions for vector operations.

Functions:

1. *Distance (vector<float>, vector<float>)*. Takes in two vectors and returns their distance.

2. *Angle (vector<float>, vector<float>, vector<float>)*. Takes in three vectors. The second vector is treated as a vertex, and the angle between the three vectors is returned.
3. *Scale (vector<float>, float) (vector<float>, int)*. Takes in a vector and a scalar. The vector is scaled according to the scalar value.

4. *VectorSum (vector<float>, vector<float>)*. Takes in two vectors and returns the resultant vector.

5. *DotProduct (vector<float>, vector<float>)*. Takes in two vectors and returns their dot product.

6. *FindLargestNegativeInteger (vector<int>)*. Takes in an integer vector and returns the index of the largest negative integer.

7. *Translate (vector<float>, int) (vector<float>, float) (vector<int>, int) (vector<int>, float)*. Takes in a vector and a scalar. The vector is translated according to the scalar value.

8. *CenterOfMass (vector<vector<float>>, vector<float>)*. Takes in a 2D vector and a 1D vector. Both inputs should have the same length. The 1D vector contains the masses of the vectors in the 2D vector. The function returns a vector for the center of mass.

9. *IsIn (int, vector<int>) (string, vector<string>)*. Takes in an int or a string and an int vector or a string vector. The function determines whether the int/string is in the int/string vector. Returns a boolean value.

10. *Where (string, vector<string>)*. Takes in a string and a string vector. Returns the index of the string in the string vector.

11. *VectorSummation (vector<float>)*. Takes in a vector and returns the sum of the values in the vector.

12. *VectorError*. Displays an error message for vector operations.

**C. main.cpp.** Instantiates a *Describe* class taking input from the stream, and calls all functions from the *Describe* class.

## III. FUTURE PLANS

The program and its documentation was written in less than a week while the author was still learning C++, so it is not at its optimum state in terms of time and space. Most functions were implemented brute-force without referring to existing functions in the standard library. First action point is to clean the code to make it run faster and manage memory more efficiently.

This program is envisioned to be further extended to support a wider selection of structural descriptors. Parsers of other trajectory file formats (CHARMM dcd, binpos, etc) and parameter file formats (CHARMM psf, etc.) will also be implemented as needed.

**References**

1. Borry Maxime. (2017). ProtInter: Protein Interaction Calculator.
2. Cock P. J. A., Antao T., Chang J. T., Chapman B. A., Cox C. J., Dalke A., Friedberg I., Hamelryck T., Kauff F., Wilczynski B., Hoon D., and L M. J. (2009). Biopython: freely available Python tools for computational molecular biology and bioinformatics. *Bioinformatics, 25(11): 1422-1423.*
3. Tina K. G., Bhadra R., and Srinivasan. (2007). PIC: Protein Interactions Calculator. *Nucleic Acid Research, Vol. 35, Web server issue*.
4. Unidata Program at the University Corporation for Atmospheric Research. NetCDF. Retrieved from http://www.unidata.ucar.edu/software/netcdf/.