Bar Weiner

Stanford Structural Biology

August 17, 2019

Final Lab Report

**Abstract:**

The field of Biology’s intersection with computer science plays an essential role in understanding the structure, interactions, nature, and dynamics of proteins. During my four week internship at the Levitt Lab of Computational Structural Biology working under Dr. Ivan Ufimtsev, I undertook a deep dive into the world and tools of research and academia within the field’s rigorous study of proteins and their nature. As I began to grasp the various problems, purpose of computation within the discipline of biology, and the almost limitless applications and doors that computers open in the field, I began shaping my coding knowledge and acclimating it into the new applications and interfaces which I would interact with to take on my projects. Over time I entered into an exploration of the fundamentals of protein structures, and how we can predict their structure and atomic coordinates from experimental crystal diffraction experiments, and alternatively determine their structure and electron locations using Discrete Fourier Transforms. These trials lead to an even further exploration of protein structure, especially an analysis of protein structure as it is broken and altered from its indigenous state as it comes into contact with multiple different aggravators and stimuli. Further tests of various software include testing Protein positioning software intended to fix the structure of broken proteins, as well as testing the impact and effectiveness of Sidechain prediction software.

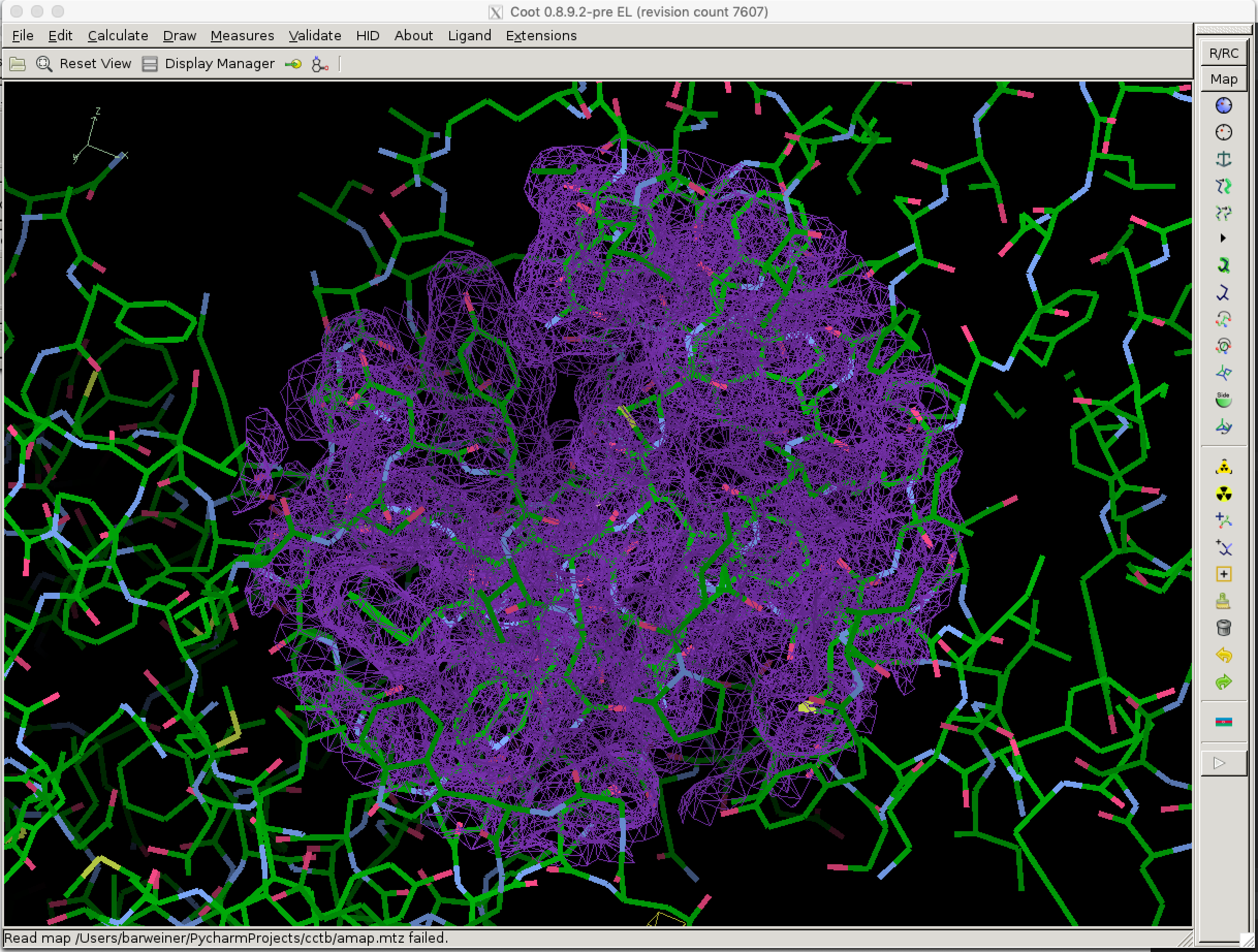
**Results and Methods:**

**Introduction**

To inaugurate my time in the lab, I was promptly introduced to the underlying problems which we were striving to find solutions for. Through detailed explanations and visual aid, I began to comprehend the various different problems which arise when dealing with concepts such as light diffraction and picks, protein Crystals and crystallography, waves as they pertain to Fourier Transforms, and different measurements of agreement and refinement between X-Ray diffraction data and the crystallographic model. Having gained this basic knowledge I proceeded to set myself up to start working with and delving into the complex nature and full extent of these concepts. I chose to install Pycharm-an integrated development environment which I utilized to compile and run my python scripts in an effective matter. This was a big leap for me in regards to my python computing. Before, whenever I developed programs in python I would succumb to writing out my code in text files which were ineffective and did not provide me the tools to tackle more rigorous coding tasks. As I downloaded and began using this new platform I also began teaching myself how to utilize my Terminal command line in order to aid me throughout my research. As I learned its many different commands and uses I began to obtain a stronger grasp on my own machine, allowing me to complete local processes and tasks within my Macbook with ease. In this process, I additionally created a Github account which would allow me to version control my projects, and collaborate with other developers within the Structural Biology field, while employing their work, libraries, and methods in my own projects. To utilize my newly created Github account I also invested time to learn the various commands and functions which Github provided. By learning to control and use the language of Git I could safely manage my projects, access their history and store their information in various repositories of my choosing.

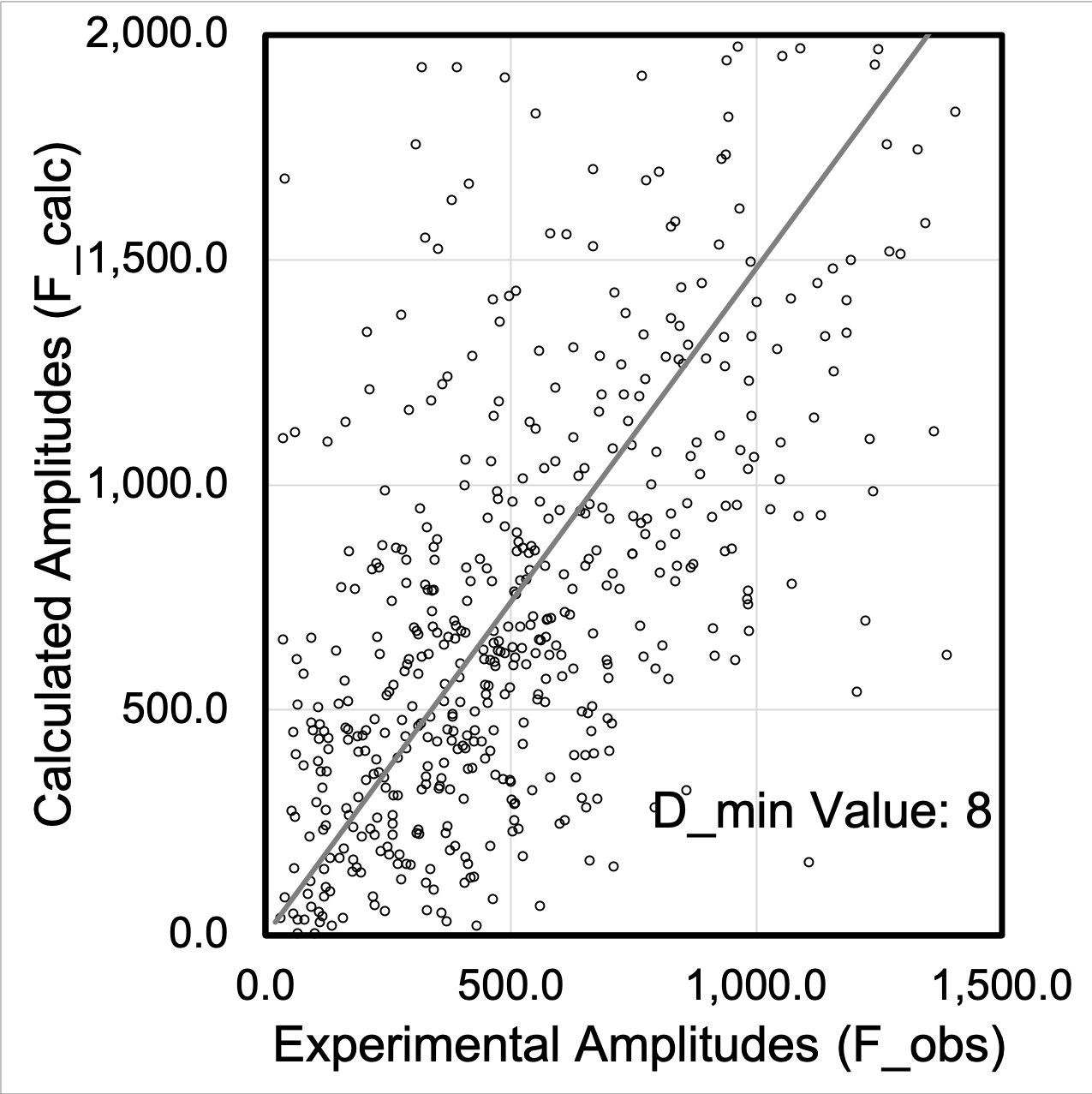
**Utilizing the Computational Crystallography Toolbox**

At this point, after acquiring the rudimentary knowledge and tools to begin working, I took steps to familiarize myself with a large Computational Crystallography library known as CCTBX with the goal of finding and utilizing methods and classes to represent and store the vast amount of data present within a protein Crystallographic Information File (CIF)- a standard text file format for representing crystallographic information. This library contains tools of all sorts to represent every type of procedure and to extract and reason with many of the data types known to Structural Biologists. After hours of searching and trying to understand how the data for these CIF files is parsed, extracted and stored, I located and utilized compatible methods to store the different values within various CIF reflection files which I downloaded from RCSB PDB- a protein data bank. I would store these values in data structures known as “Miller Arrays” which I could later access. Through these data types, I could encapsulate the various values which were stored in the gargantuan CIF files, and access just the array indexes which stored the wave amplitudes, and the atomic coordinates which I would later use in my calculations. At this point I repeated a similar process, scavenging through the CCTBX library and its corresponding extensive documentation in order to locate the correct procedure for extracting data from a different file which stored Protein data known as PDB files- a database for the three-dimensional structural data of large biological molecules, such as proteins and nucleic acids. When researching how to achieve my goal of extracting the data from these file formats, I learned of the hierarchical system of data storage for atoms as they fit under families of models, chains, and residues of proteins. I later began to utilize my knowledge about the location of atomic coordinates within this large hierarchy of data in order to extract the necessary values from these various files. With this data, I was able to perform a series of Fourier transforms on the atomic coordinates given in the files in order to calculate the amplitudes of the waves from the data. Furthermore, using the PDB file format, I could implement and modify methods from the CCTBX library to write programs of my own which created electron density maps and unit\_cell maps that I could later visualize and compare to other variations of the protein structures using the Crystallographic Object-Oriented Toolkit(COOT) interface for Crystallography.



**Figure 1.**This figure is the electron density displayed in purple superimposed onto the zoomed in 6f0o protein structure

**Analyzing Computed and Experimental Data**

With these calculated amplitudes, which I was able to produce from the raw coordinates presented in the PDB file format, I compared my calculated values to the real experimental data which I had extracted from the X-ray diffraction data presented in the CIF reflection file.

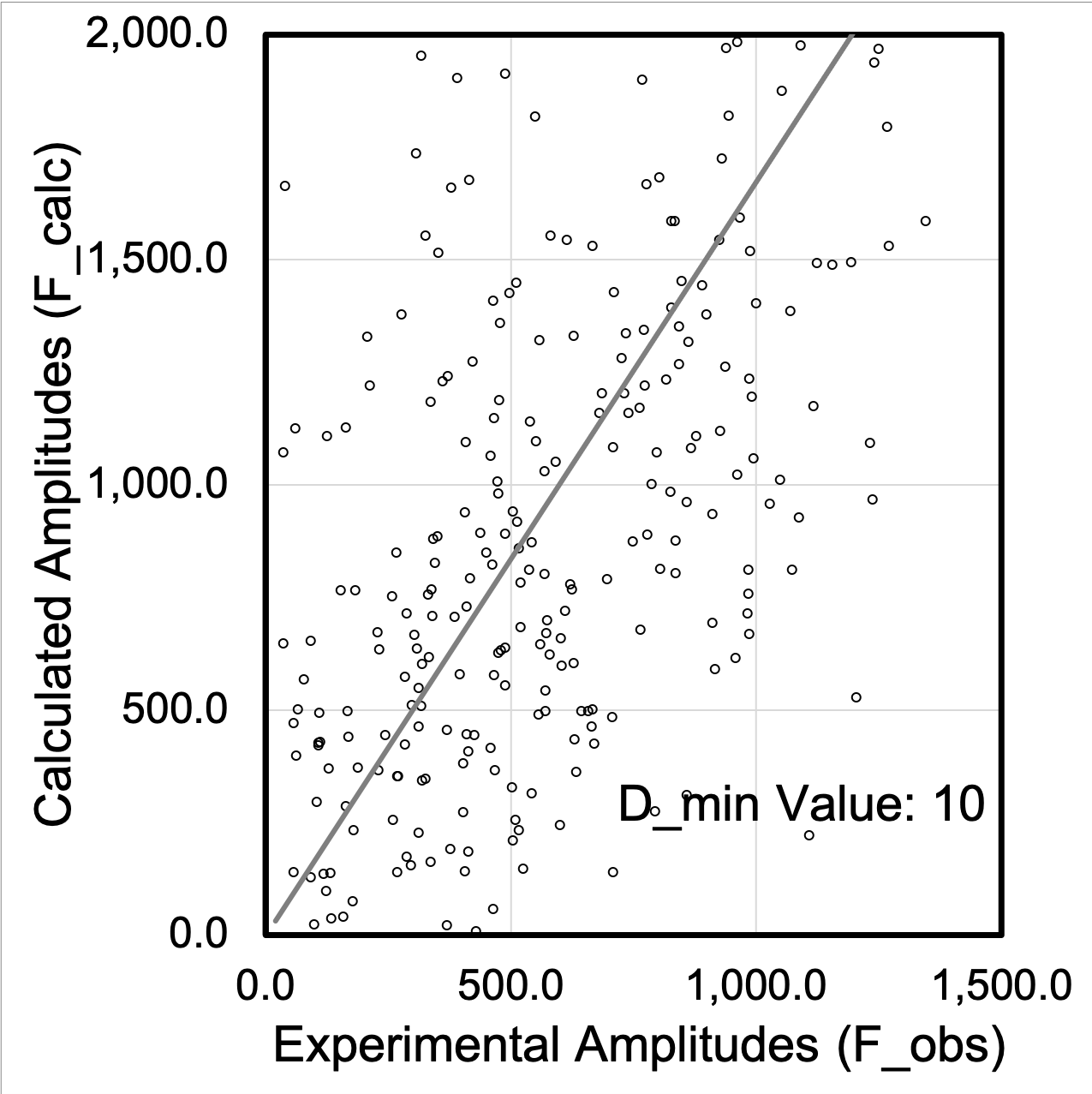
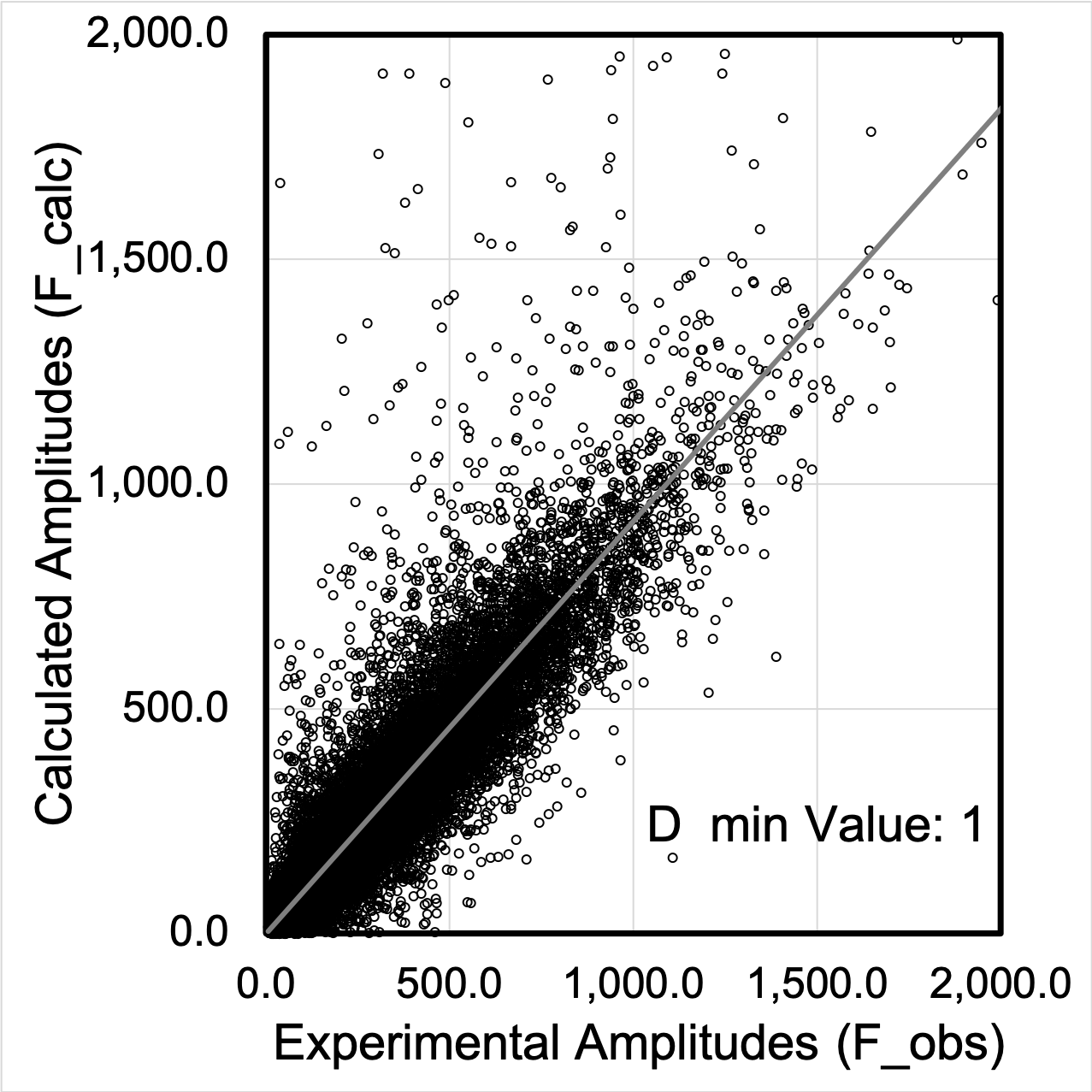
|  |  |
| --- | --- |
| f1 = open("computedValues.txt", "r")  f2 = open("theoreticalValues.txt", "r")  f3 = open(r"merge.txt", 'w')  lines = [line for line in f1.readlines()]  lines2 =[line for line in f2.readlines()]  def find\_nth\_overlapping(haystack, needle, n):  start = haystack.find(needle)  while start >= 0 and n > 1:  start = haystack.find(needle, start+1)  n -= 1  return start  for line1 in lines:  for line2 in lines2:  if (line1[line1.find('('):line1.find(')')] == line2[line2.find('('):line2.find(')')]):  String = line2[line2.find(')')+1: line2.find('.')+2],line1[line1.find(')')+1:]  String = str(String)  String = String[2:String.find(',')-1] + " " + String[String.find(',')+4:String.find(')')-4]  f3.write((String) + '\n')  break  #f2.seek(0) | import functools  def run():  f1 = open("computedValues.txt", "r")  f2 = open("theoreticalValues.txt", "r")  f3 = open(r"merge.txt", 'w')  def find\_nth\_overlapping(haystack, needle, n):  start = haystack.find(needle)  while start >= 0 and n > 1:  start = haystack.find(needle, start+1)  n -= 1  return start  def compare(item1, item2):  if (float(item1[item1.find('(') + 1: item1.find(',')]) > float(item2[item2.find('(') + 1: item2.find(',')])):  return 1  elif(float(item1[item1.find('(') + 1: item1.find(',')]) < float(item2[item2.find('(') + 1: item2.find(',')])):  return -1  elif(float(item1[item1.find(',')+2:find\_nth\_overlapping(item1,",",2)]) > float(item2[item2.find(',')+2:find\_nth\_overlapping(item2,",",2)])):  return 1  elif(float(item1[item1.find(',')+2:find\_nth\_overlapping(item1,",",2)]) < float(item2[item2.find(',')+2:find\_nth\_overlapping(item2,",",2)])):  return -1  elif (float(item1[find\_nth\_overlapping(item1, ",", 2) + 1: item1.find(')')]) > float(item2[find\_nth\_overlapping(item2, ",", 2) + 1:item2.find(')')])):  return 1  elif(float(item1[find\_nth\_overlapping(item1, ",", 2) + 1: item1.find(')')]) < float(item2[find\_nth\_overlapping(item2, ",", 2) + 1:item2.find(')')])):  return -1  elif(item1[item1.find('('):item1.find(')')] == item2[item2.find('('):item2.find(')')]):  return 0  else:  return -1  lines = [line for line in f1.readlines()]  lines2 =[line for line in f2.readlines()]  lines.extend(lines2)  total = sorted(lines, key=functools.cmp\_to\_key(compare))  for i in range(len(total)-1):  line1 = total[i]  line2 = total[i+1]  if(line1[line1.find('('):line1.find(')')] == line2[line2.find('('):line2.find(')')]):  String = line2[line2.find(')') + 1: line2.find('.') + 2], line1[line1.find(')') + 1:]  String = str(String)  String = String[2:String.find(',')-1] + " " + String[String.find(',')+4:String.find(')')-4]  f3.write((String) + '\n') |

A B

**Figure 2.** These are the python scripts I used to merge the experimental data, and the computed data into one document with two columns of amplitudes, one for each file, that I could quickly transfer into excel. Table A shows my initial naive attempt of looping through each file in nested for loops with an O(n^2) runtime which took 35 minutes for each merged file to finish. On the other hand table B shows my complete implementation which had an O(nlog(n)) runtime and completed the task in 12 seconds on my machine.

In the process of extracting data and analyzing it through multiple different lenses, the key to making my projects work as effectively as possible was efficiency. Run time efficiency and time to run my programs played a big part in how I designed and implemented my code. Due to the sheer size of many of the files which I drew data from, it was extremely impractical to run anything above the minimum time complexity. Thus, whenever designing my programs I meticulously thought out and optimized my methods to perform optimally when running large files containing the information regarding my proteins such as in example B of Figure 2.

When comparing the data, I utilized Microsoft Excel spreadsheets and scatter plots to graph our findings on how the relationship between the calculated amplitudes and the observed amplitudes changed as the D\_min value fluctuated in my calculation of the amplitudes from the PDB files. We hypothesized that as the D\_min values decreased, an increasing linear relationship would form between our calculated amplitudes and the experimental amplitudes due to the increase in data points caused by a lower D\_min. This was confirmed by our data in the following charts:



A B

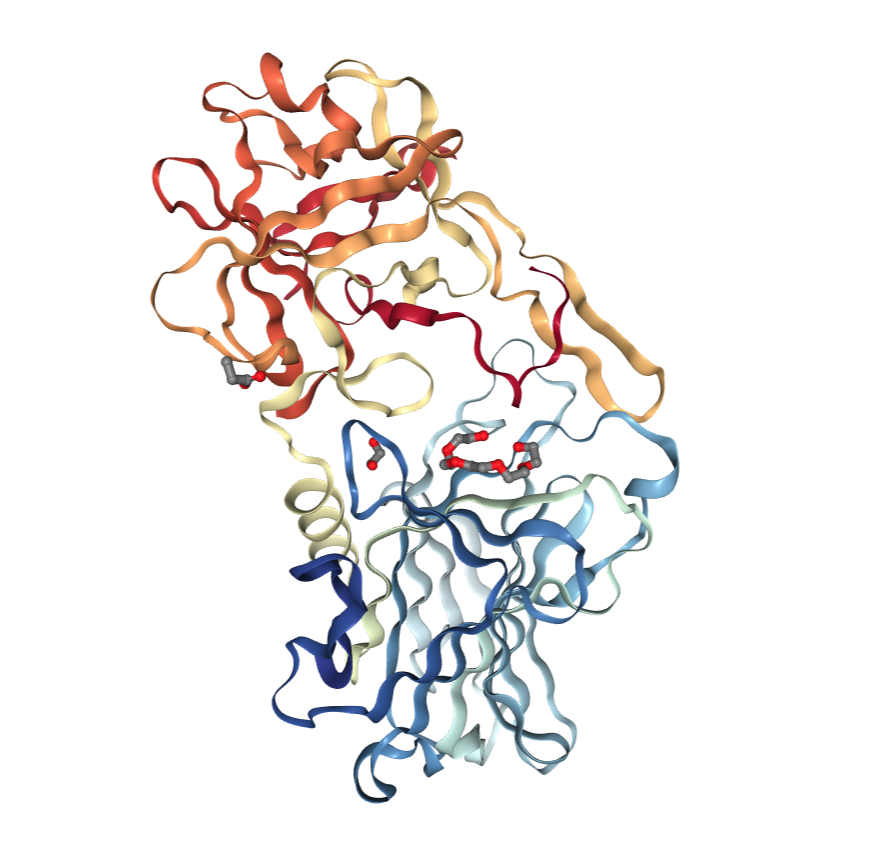
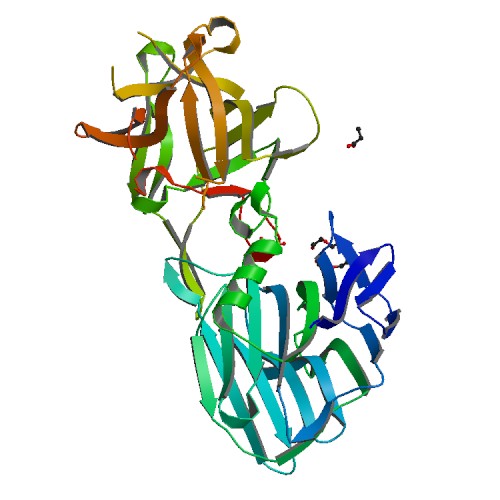
**Figure 3.** Graph A shows the graph relating the calculated amplitudes and the experimental amplitudes at the lowest D\_min we tested at value 1, graph B shows the same relationship at the largest tested D\_min at the value of 10.

Analysis of these graphs confirms our hypothesis as they display that as the D\_min value gets higher, the relationship between the calculated amplitude data and the experimental amplitude data becomes less linear. At the lower D\_min values there are more points overall because the distance in Angstrom(Å) between each of the points being detected is smaller. In other words, as the D\_min is smaller, we split our constant segment of the crystal structure into more data points, which are each closer together because the frequency of the data points got higher. This increase in data points provides a larger sample size and helps balance out some of the possible errors and inaccuracies within the Discrete Fourier Transform calculation process and creates a more linear trend throughout the entirety of the data set, as the calculated points are compared to the experimental points.

**My Model Protein**

Throughout the process of extracting data and doing calculations which reveal patterns and trends regarding the structure of proteins, I conducted most of my tests on a protein called 6f0o. This protein(6f0o) often known as Botulinum neurotoxin A3 Hc domain fascinated me from the beginning due to its applications into various aspects of our integrated society in many domains despite its apparent lethality even in microscopic doses. This protein prevents the release of the neurotransmitter acetylcholine from axon endings at the neuromuscular junction and thus causes flaccid paralysis, yet it is still employed in medicine, cosmetics and research. For example, within the medical field this protein has been used to treat various muscle spasms and diseases characterized by overactive muscles while also being tested to treat excessive sweating and chronic migraines. Similarly in the cosmetic field this protein has been marketed under the procedure known as Botox- an injection into the face effective for reduction of facial wrinkles and signs of aging. Thus, in order to keep my calculations accurate, I chose 6f0o to serve as a constant protein structure in most of my tests.

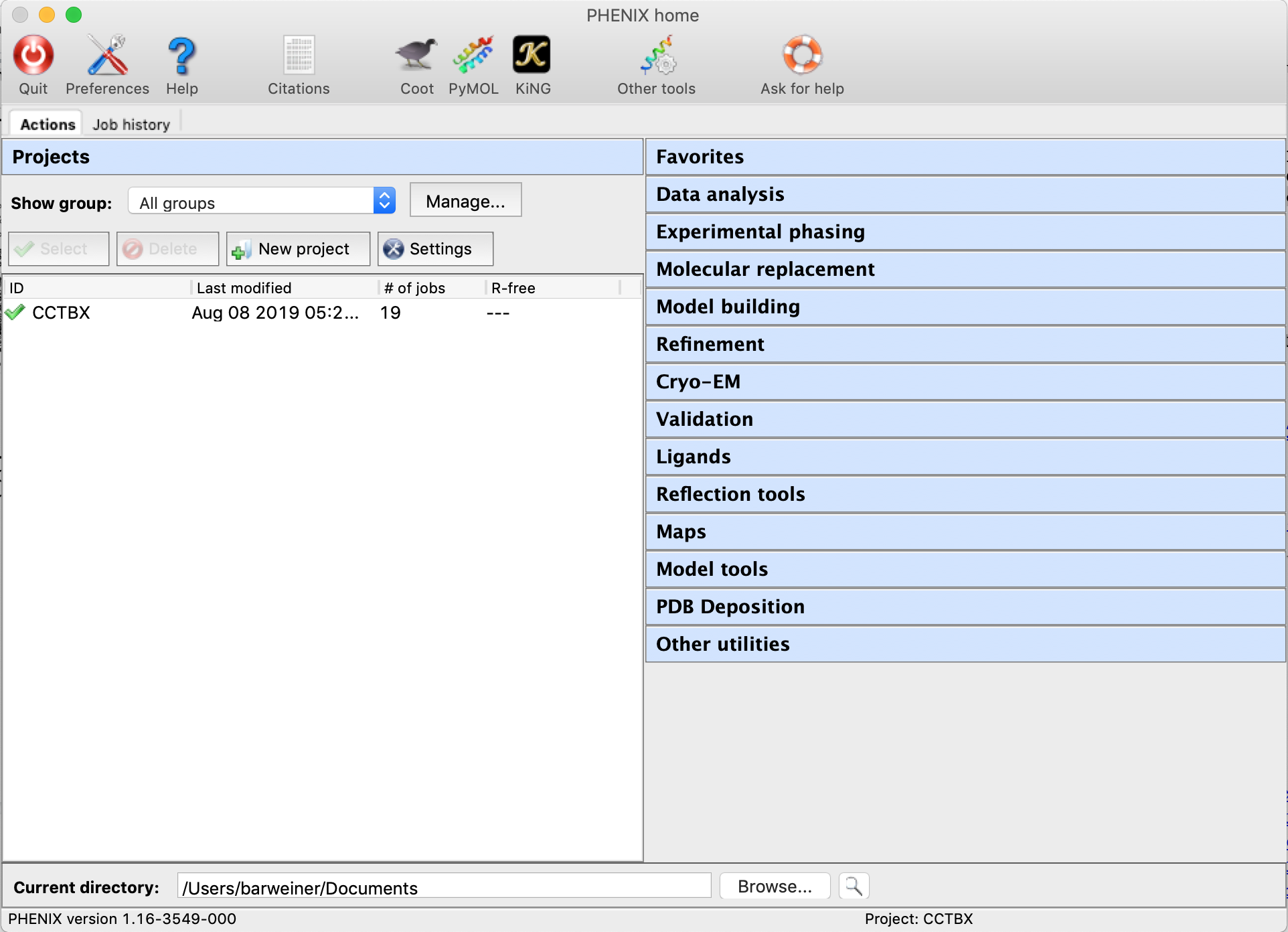




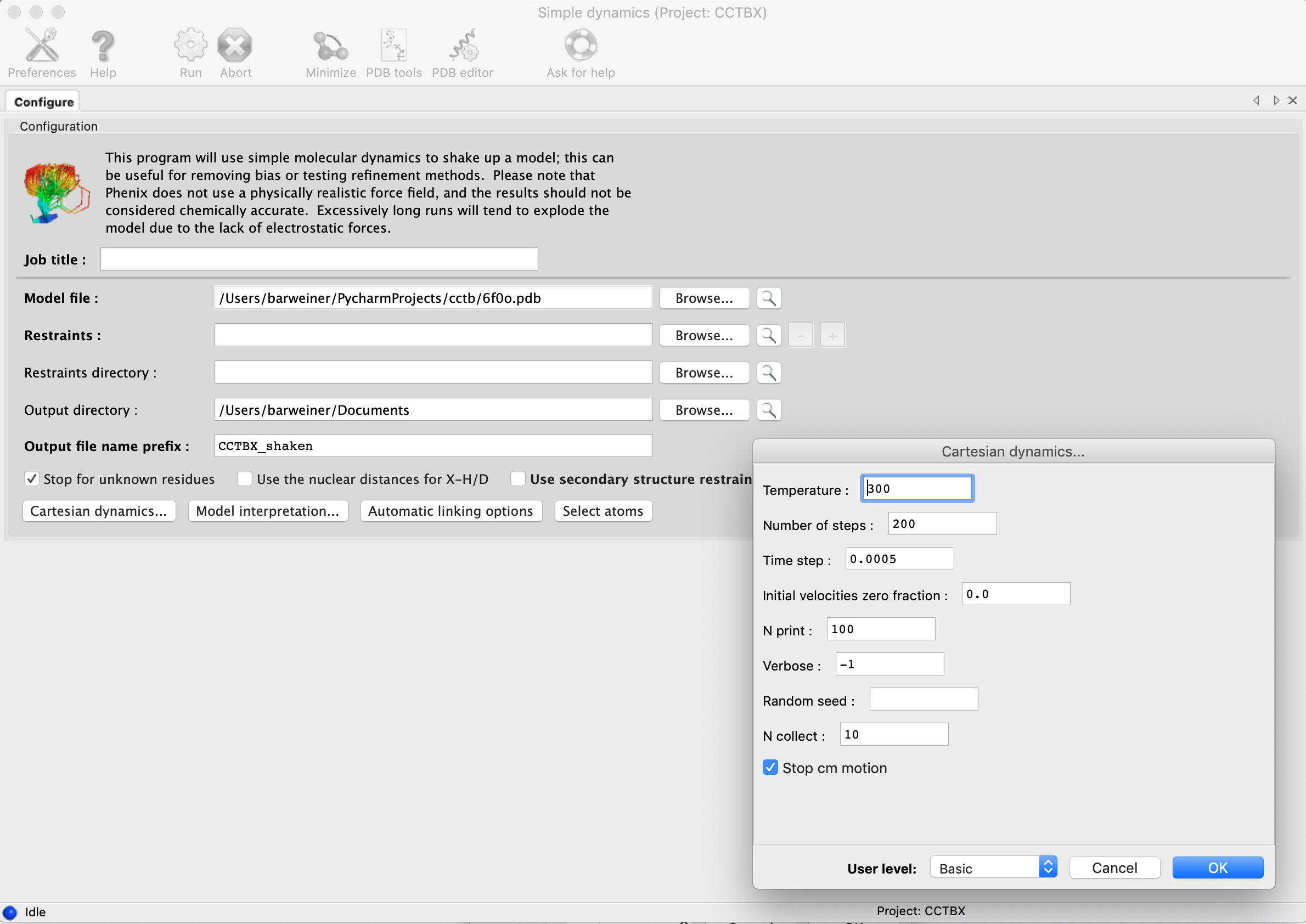
**Figure 4.** The protein structure of the Botulinum neurotoxin A3 Hc domain is displayed for reference.

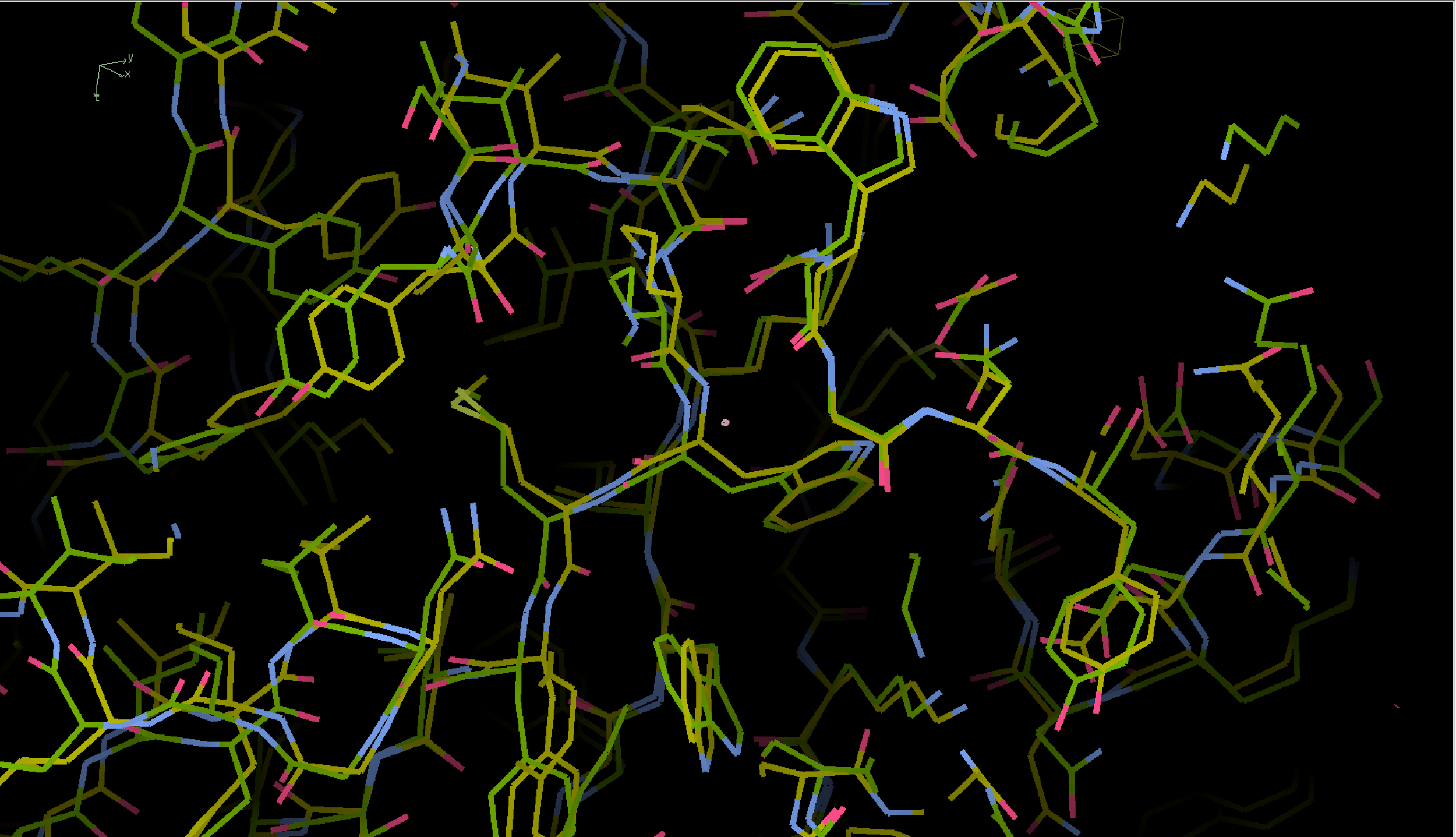
**Breaking protein structures**

At this point, our next project and exploration into the structure of proteins revolved around the impact of “breaking” the structure of the protein to various extents by short molecular dynamics simulations trials, while comparing them back to the model as presented in the PDB files. In order to break our protein models I installed a program called Python-based Hierarchical ENvironment for Integrated Xtallography(phenix) seen in figure 5, an interface that allows the user to simulate a protein’s structural change as it becomes exposed to various different conditions such as temperature change, while giving discretion over the number of steps taken to simulate the molecular dynamics.

We could then go onto visualize these small structural changes, carried out by the phenix simple dynamics algorithm, in COOT as we visualize our two proteins side by side, both centered on the screen and zoomed in on the atomic level(Figure 7). 

**Figure 5.** Above is the Phenix overarching interface which I used for my research



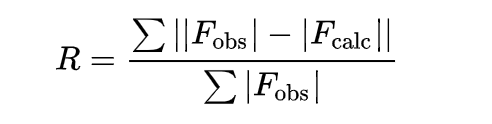
**Figure 6.** This image displays the Simple Dynamics interface which I utilized to shake up and break my model protein’s structure. On the right is the cartesian dynamics option menu which I used to systematically conduct various trials and simulations on my protein’s structure.

**Figure 7.** This zoomed in model illustrates the small shift in the atomic coordinates and positioning between the two proteins as represented in the image by the yellow chain, and the green chain.

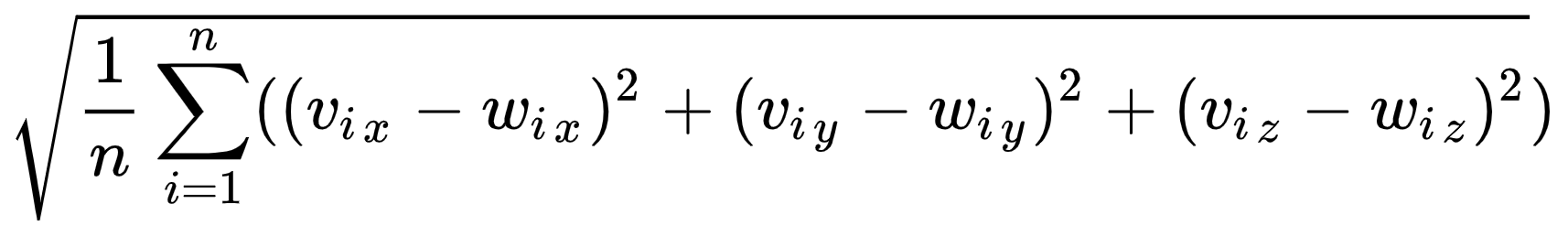
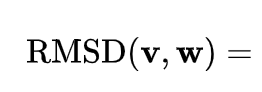
**Mathematical Analysis of Similarity**

Furthermore, beyond visual overlay of the broken protein and the model, we employed data analysis and crystallographic calculations to make sense of and understand the new data we had created through the simulations. One of the primary comparisons we made was through the measure of agreement between the crystallographic model and the experimental X-ray diffraction data in a calculation known as the Residual factor(R-factor). In this series of computation seen in Table 9A , the R-factor was calculated by passing in the amplitudes and coordinates of the experimental data and the calculated data from the broken protein structures into a python script which utilized the R formula(1).

The other calculation we initiated to compare the model protein with our newly broken structure was the Root-mean-square deviation(RMSD) of atomic positions, which is the measure of the average distance between the atoms of our two superimposed proteins as seen in Table 9B. We calculated the RMSD by running the data from the broken protein data and the model protein data through a python program which utilized the RMSD equation(2).



(1)



(2)

To provide adequate sample structures to provide a foundation for the calculations, we used phenix software to provide ample distinct data points and distinct molecular formations. Using phenix(figure 8) we randomized the number of steps our series of breaks would take on to deviate from our model protein. We output these simulations as hundreds of individual broken Protein files which we could later access and extract the new structures’ composition from.

|  |
| --- |
| **#!/bin/bash**  for ((i=1;i<1000;i++)); do  ranNum2=$[RANDOM%300+9700]  phenix.dynamics /Users/barweiner/PycharmProjects/cctb/6f0o.pdb temperature=500 steps=$ranNum2 output\_file\_name\_prefix=$i$\_shaken write\_geo\_file=False  #Python3 /Users/barweiner/PycharmProjects/cctb/AllOperations.py 6f0o$i.pdb  done |

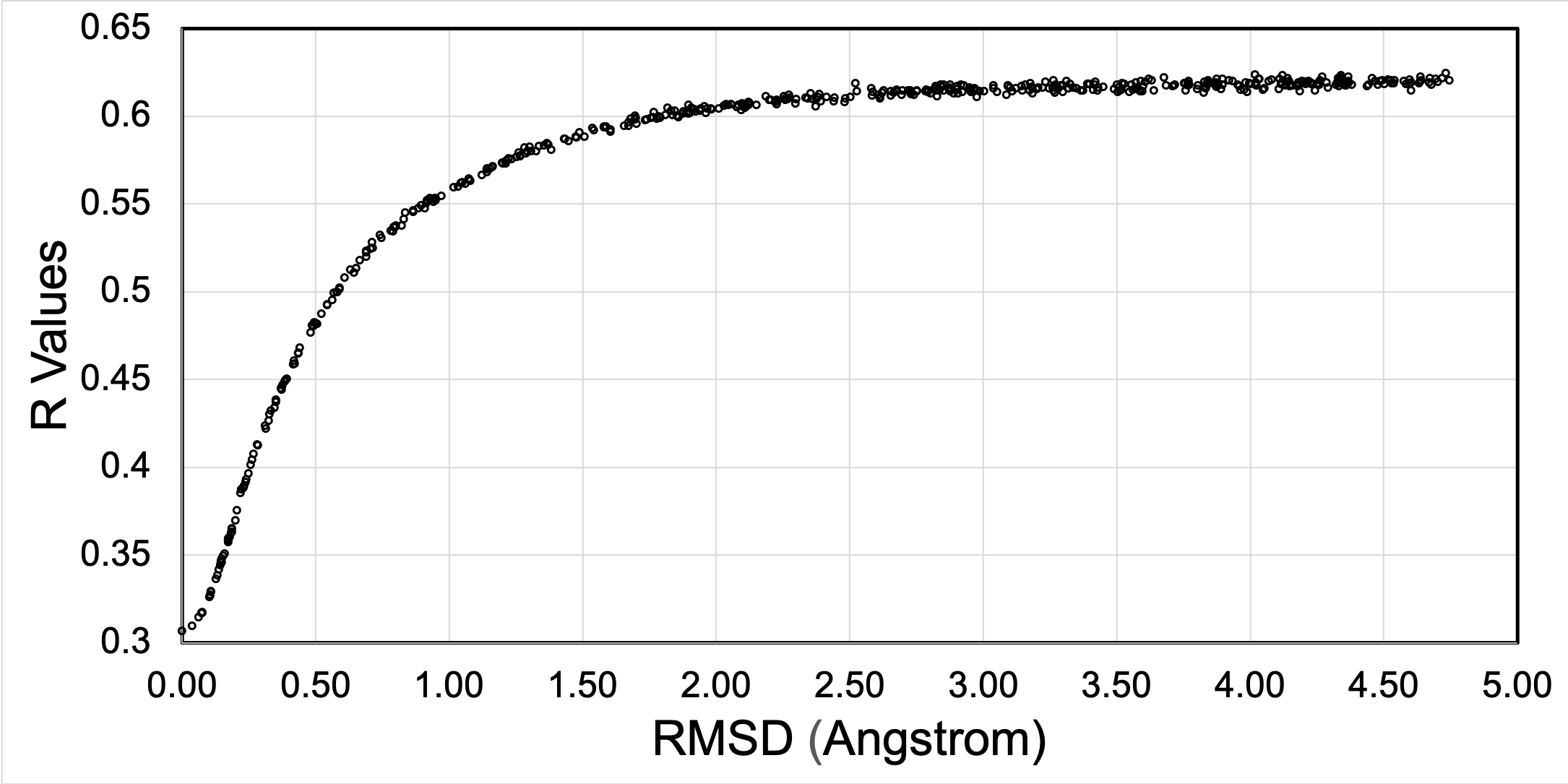
**Figure 8**. The bash script above was used to simulate thousands of proteins using a random amount of dynamic motion steps. It then saved their structures to unique files on my local machine.

|  |  |
| --- | --- |
| def find\_nth\_overlapping(haystack, needle, n):  start = haystack.find(needle)  while start >= 0 and n > 1:  start = haystack.find(needle, start+1)  n -= 1  return start  def calc\_RMSD(fileName):  f1 = open("6f0o.pdb", "r")  f2 = open(fileName, "r")  lines = [line for line in f1.readlines()]  lines2= [line for line in f2.readlines()]  DifferenceSum = 0  counter=0  for i in range(len(lines)):  if(lines[i].find(" CA ")!= -1):  counter = counter +1  DifferenceSum = DifferenceSum + ((float(lines[i][(lines[i].find(".")-3):(lines[i].find(".")+4)]) - float(lines2[i][lines2[i].find(".")-3: lines2[i].find(".")+4]))\*\*2)  DifferenceSum = DifferenceSum + (((float(lines[i][find\_nth\_overlapping(lines[i], ".", 2) - 3: find\_nth\_overlapping(lines[i], ".", 2) + 4]) - float(lines2[i][find\_nth\_overlapping(lines2[i], ".", 2) - 3: find\_nth\_overlapping(lines2[i], ".", 2) + 4])) \*\* 2))  DifferenceSum = DifferenceSum + (((float(lines[i][find\_nth\_overlapping(lines[i],".",3) -3: find\_nth\_overlapping(lines[i],".",3)+4]) - float(lines2[i][find\_nth\_overlapping(lines2[i],".",3) -3: find\_nth\_overlapping(lines2[i],".",3)+4])) \*\* 2))  DifferenceSum = DifferenceSum/counter  DifferenceSum = (DifferenceSum) \*\* (0.5)  return DifferenceSum  f3 = open(r"RMSD\_Values.txt", 'w')  for i in range(95):  RMSD\_Value = calc\_RMSD  f3.write(str(RMSD\_Value) + '\n') | def calc\_R():  f1 = open("merge.txt", "r")  Difference= 0  Sum=0  for i in f1:  #print(float(i[0:i.find(".")+1]))  Sum = Sum + float(i[0:i.find(".")+1])  f1.seek(0)  for j in f1:  num1= float(j[0:j.find(".")+1])  num2= float(j[j.find(".")+3: ])  Difference = Difference + abs(num1-num2)  rFactor = Difference/Sum  return rFactor |

A B

**Figure 9.** Table A shows the script I used to calculate the RMSD of each of my broken protein structures. Table B is the python code I used to calculate the R value for each file.

On each of our newly created files, we calculated the RMSD and the R Values so that we could compare them for each individual broken protein. (Table 9) To calculate the R Values we first had to extract the amplitudes and atomic coordinates out of each broken protein PDB file using discrete fourier transforms. We then could utilize the Residual(R) value as a data point for each protein structure. We visualized this data relating the protein’s R value and RMSD value using an excel scatter plot(Figure 10).

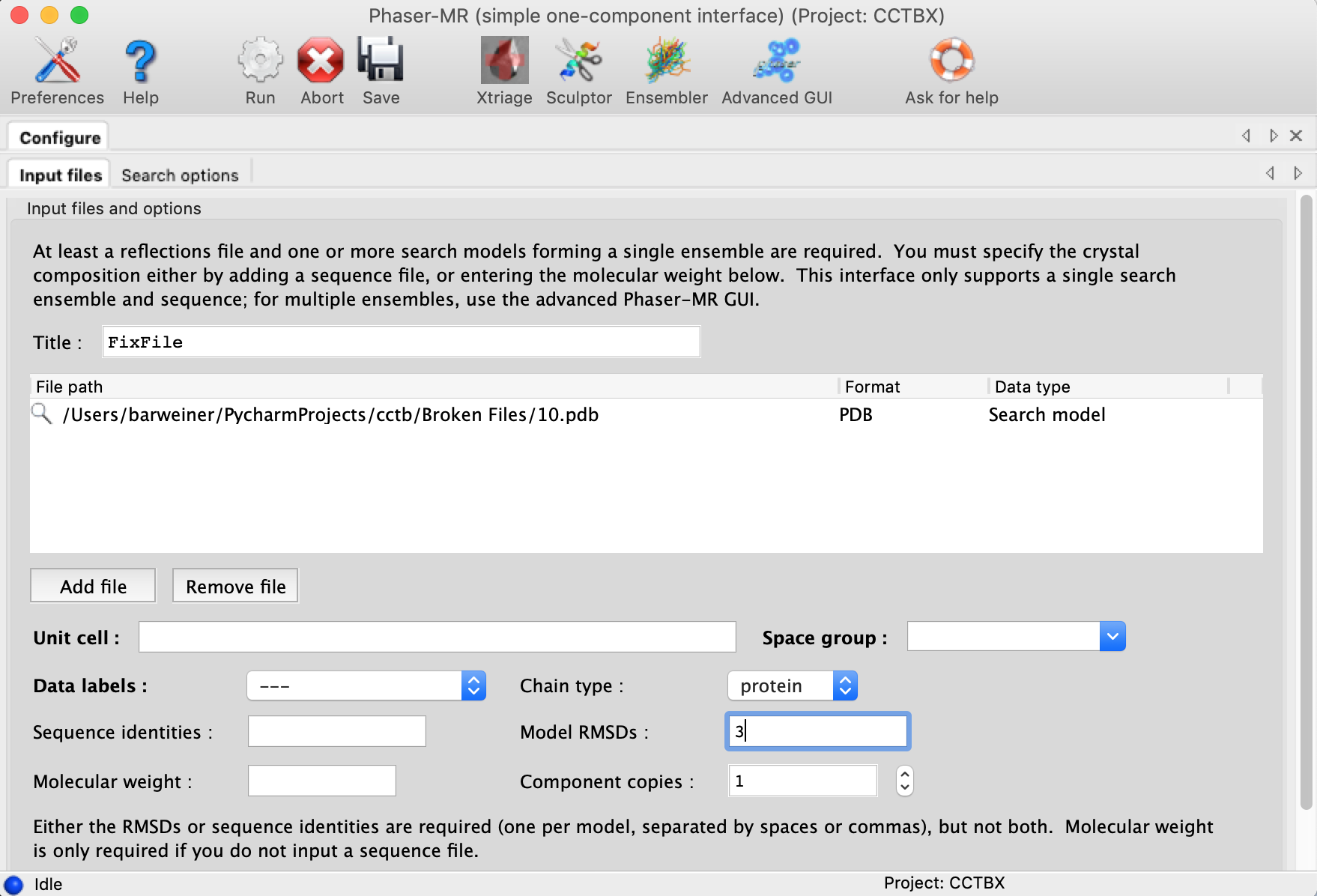


**Figure 10.** The R Value vs RMSD(Angstrom) of each broken, simulated protein structure.

**Further Unsuccessful Attempts at Tampering with Protein’s structure**

Later, after collecting the data from the calculations of R Values and RMSD values taken from our broken protein simulations, we tried to solve our molecule even more effectively by utilizing phenix’s Phaser feature(Figure 11) which uses automated molecular replacement to find ideal locations for atoms so that the model to explain the diffraction data. However, over multiple attempts using the phaser program, it was consistently not able to produce a structure that fit the model better than the protein structure that was broken through the simple dynamics simulation. When passing in broken structures, we expected a file which produced an R value and RMSD that were less than the initial broken structure because the phaser is intended to provide improved coordinates in order to position the proteins effectively for crystallization. Nevertheless, when testing structures, the phaser created atomic structure models which had even higher RMSDs and inflated R values.

With the same goal in mind as the simple dynamics simulator, in order to further break our structure we tried to utilize the library SCWRL4- a protein side chain predicting algorithm. By using this program we had hoped to break the protein further, by counting on the inaccuracies of the program to negatively impact the overall structure when we solely entered in the backbones of protein structures. However the libraries method of reinstating all of the side chains was tested to be very effective, as the new protein structure with the externally predicted side chains had an R value very similar to the original broken protein which we stripped of its sidechains and entered into SCWRL4.



**Figure 11.** Above is the Phenix Phaser which we tried to use to fix our broken simulated protein structures

**Conclusions:**

Overall in my four weeks in the lab, I gained insight into the various tools and into the ordinary field and structure of work as a structural biologist. I learned how to utilize various programs, software and interfaces to predict and analyze data from proteins using crystallographic calculations and observations, while experimenting and learning how to display my results in an effective manner. On a larger level, I learned how to balance accuracy and efficiency in my program designs, and gained an understanding on dealing with and handling large data sets effectively using the tools present on my machine. After bringing this 4 week internship to a close, I think that with the various skills I have aquired I am ready to take on more, and continue exploring the different ways to analyze large sets of data and make inferences and predictions in the Structural Biology field, and in the rest of my life. These new lessons that I will build on over time, set me up to continue on my path as a scientist with the goal of bettering humanity, and of deepening our understanding of the world around us.

**References:**

1. R. J. Gildea, L. J. Bourhis, O. V. Dolomanov, R. W. Grosse-Kunstleve, H. Puschmann, P. D. Adams and J. A. K. Howard: iotbx.cif: a comprehensive CIF toolbox. J. Appl. Cryst. (2011). 44, 1259-1263.
2. Nature Structural Biology 10, 980 (2003)
3. Emsley, Paul, et al. “Features and Development of Coot.” Acta Crystallographica Section D - Biological Crystallography, vol. 66, 2010, pp. 486–501.
4. Davies, J.R., Rees, J., Liu, S.M., Acharya, K.R.(2018) J. Struct. Biol. 202: 113-117
5. P.D. Adams, P.V. Afonine, G. Bunkoczi, V.B. Chen, I.W. Davis, N. Echols, J.J. Headd, L.W. Hung, G.J. Kapral, R.W. Grosse-Kunstleve, A.J. McCoy, N.W. Moriarty, R. Oeffner, R.J. Read, D.C. Richardson, J.S. Richardson, T.C. Terwilliger, and P.H. Zwart. Acta Cryst. D66, 213-221 (2010).
6. G. G. Krivov, M. V. Shapovalov, and R. L. Dunbrack, Jr. Improved prediction of protein side-chain conformations with SCWRL4. Proteins (2009).
7. Kufareva, Irina, and Ruben Abagyan. “Methods of Protein Structure Comparison.” Methods in Molecular Biology Homology Modeling, 2011, pp. 231–257., doi:10.1007/978-1-61779-588-6\_10.
8. “R Factor.” R Factor - Online Dictionary of Crystallography, http://reference.iucr.org/dictionary/R\_factor.

**Supplemental Materials:**

