Multiscale Modeling of Amino Acids

A Major Qualifying Project

submitted by

Anders Hokinson

in **May 2013**

to the faculty of Worcester Polytechnic Institute, Worcester, MA,

in partial fulfillment of the requirements for the

degree of Bachelor of Science in the department of

Physics

and accepted on the recommendation of

Prof. Izabela Stroe, Ph.D.

Department of Physics

&

Kryngle Daly, Ph.D.

KBioSim

Protein misfolding results in a wide range of highly debilitating and increasingly prevalent diseases. Despite success in atomic protein folding simulations, timescales for results are long and hinder research into the stochastic folding process. Performing Langevin dynamics simulations with NAMD on amino acids shows that atoms within the amide and carboxyl groups remain rigid enough to suggest motion of the amino acid can be characterized by these two groups. Ramachandran plots verify this when compared to those from experimental literature for glycine. Early results suggest that it is possible to reduce the number of computational elements in protein folding simulation by rescaling the physical analysis to the atom groups.

Key Words: Protein Folding, Amino Acid, Model, Multiscale Physics, NAMD, Simulation, Ramachandran Plot, Glycine

In addition to my advisors, Professor Izabela Stroe and Kryngle Daly who have shared with me the insight and ability to complete my work, I would also like to thank the entire Physics Department at WPI, whether or not we have worked together directly.

I would specifically like to thank Professor Erkan Tüzel for introducing me to computational physics early in my studies and providing me with the background to undertake this project.

I would like to thank my parents who have helped me through everything, every step of the way.

My friends, my true friends, thank you.

Arielle, you were my muse. You have been and always will be my best friend.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

A biophysical approach to computational biology creates two important barriers which must be overcome for progress of the physical understanding of biological systems. The level of abstraction used in representing the molecule and the forces describing interactions bring about limitations in the biological object that is being represented. The other issue lies in the sampling, in that the length of a simulation can be influenced by the different configurations of the biological object that can be visited. This leads to questioning how much interaction between objects is truly involved.

Limitations in sampling can be directly attributed to the limits of current computers, which are typically 1,000 to 100,000 times too slow for the demands of modern computational biology[9]. Even using the best algorithms for specific computational biology experiments, the simulation may still be too long (on the order of several months) and thus conclusively inefficient.

As Newton was describing the motions of the planets, an atomistic model was neither necessary nor desired. For proteins, everything from lattice models, to off-lattice simplified alpha-carbon models, to fully atomistic models have been em-

ployed. None so far have overcome the limitations of these barriers.

Attempts to quantify biology on a molecular level have been hindered by the vast amount of computer power required to model the complexities of molecular dynamics (MD). The wealth of knowledge regarding genomic and proteomic data combined with advances in computational algorithms and ever growing computational power will open the door to biomedical advances which allows new predictive techniques for combatting disease from protein misfolding[9].

Still, running fully detailed, fully atomistic MD simulations is clearly not a viable computational approach to the study of protein folding. Even the fastest proteins fold on the timescale of 10's of microseconds[9]. Simulations using MD software are typically limited to the nanosecond timescale, a considerable difference. Since the nature of the protein folding problem will only be observed with longer atomic simulations, researchers must either wait for better computer architecture or develop a new technique to observe and predict the nature of protein folding and misfolding.

Thus, simulating protein folding using full atomic interactions is computationally extensive and a new approach is necessary to quickly and efficiently predict and determine the conformation of proteins. This will allow researchers to better understand the mechanisms for protein folding applications in the biomedical field. This paper investigates the possibility of a new method for determining protein folding by developing models at the scale of amino acid structures, rather than the individual atomic interactions. The method aims to reduce the amount of objects in the computation making protein folding simulation less computationally expensive, thereby decreasing simulation time.

Chapter 2

Literature

2.1 Protein Folding

The folding of proteins into their compact three-dimensional structure is the most fundamental and universal example of biological self-assembly. Understanding this complex process will provide unique insight into the way in which a biological system develops its functionality[6]. The wide variety of highly specific structures that result from protein folding determines diversity in the underlying chemical processes they perfrom[6].

Only correctly folded proteins are able to interact as intended along their metabolic pathways. Despite plenty of safe-guards, given the enormous complexity and the stochastic nature of the folding process, it would be remarkable if misfolding never occurred[6]. Aggregration of misfolded proteins that escape the cellular regulatory mechanisms is a common feature of a wide range of highly debilitating and increasingly prevalent diseases such as Alzheimer's disease, Parkinson's disease, and Type-II Diabetes[6].

Native states of proteins almost always correspond to the structures that are most

thermodynamically stable under cellular conditions. Despite this, the total number of possible conformations of any protein is so large that a systematic search for a particular structure takes an incredible amount of time. Even worse, the folding process involves a series of steps between specific partly-folded states, a search of the many conformations accessible to a protein with amino acids continuously added to the polypeptide, where as the protein is assembled, the local conformations are affected by previous conformations[5].

The manner in which a newly synthesized chain of amino acids transforms itself into a folded protein depends both on the amino acid sequence and on multiple contributing influences within the cellular environment (e.g. pH). The folding and unfolding of proteins are crucial to regulating biological activity and targetting proteins to different cellular locations[6].

To understand the folding process, it is key to understand how the correct fold emerges from such fundamental steps. How is the conformational landscape unique to a specific protein defined by its amino-acid sequence? The structural transitions taking place during folding in vitro can be investigated in detail by a variety of techniques, ranging from optical methods to NMR spectroscopy, some of which can now even be used to follow the behaviour of single molecules including these amino acids[14].

2.2 Stereoscopic Experimentation

Compared to traditional microscopy, the electron microscope has two advantages for biological viewing. It has extremely high resolution and it has a great depth of focus. Both of these advantages allow for observation of extremely small biological objects.

Despite extremely laborious sample preparation, stereoscopic experimentation using electron microscopy allows for the studies of cells and different cell bodies. In particular, the use of these methods allows for better determination of cellular body structures. There is potential for this microscopy to help resolve the conformation of proteins.

The shapes of the proteins can be determined in detail. A picture of the specimen is taken, the specimen is tilted through a definite angle, and another picture is taken. The two pictures form a stereoscopic pair which, mounted side by side, can be viewed and fused to give the impression of depth[1]. With this depth, a two-dimensional representation may yield three-dimensional coordinate data which is an important characterization in amino acid sequences.

2.3 Ramachadran Resolving

By use of stereoscopic experimentation in biology, scientists have been able to resolve the three-dimensional structure of proteins and more importantly of individual amino acids. Despite their intricate architecture, revealed in thousands of 3D structures stored in the Protein Data Bank, protein structures rest on a surprisingly small set of principles[2]. Perhaps most fundamental of all is the fact that the amide bond is planar, so that only two dihedral angles, denoted by Φ and Ψ seen in Figure 3.2, define the conformation of the bond linking adjacent amino acids.

Following leads from their studies of the structure of collagen, the predominant protein group in mammals, the crystallographer G. N. Ramachandran and his colleagues first used a 2D diagram to depict the geometry of a dipeptide (two amino acids together) with the intervening bond[13]. Using the few peptide structures then available, they could see that when the angles were plotted against one another as

in Figure 4.5 (a), they clustered in only a few sections of the map.

Model building led them to conclude that most values of the two angles were inaccessible owing to collisions between atoms in the amino acid[15]. Thus, by observing these clusters on what is aptly named a Ramachandran Plot, specific amino acids may be identified. Each amino acid has a distinct region characterization, or a fingerprint, which is specific to the amino acid and crucial in its contribution to protein conformation.

2.4 Previous Simulation

More details of how the protein mechanism is able to generate a unique fold have emerged from a range of theoretical studies, particularly involving computer simulation techniques. Of particular significance are investigations that compare the simulation results with experimental observations[6].

2.4.1 Folding@Home

A new computing paradigm exists thanks to a worldwide distributed computing environment, consisting of hundreds of thousands of heterogeneous processors, volunteered by private citizens across the globe[9].

Folding@Home seeks to solve the protein folding problem through distributed computing. While it does not attempt any novel algorithms, by harnessing the thousands of computers throughout the world, computational barriers are lifted. Instead of each computer simulating a single protein molecule, the folding of a number of molecules occurs in many parallel simulations and the first simulation to cross the free energy barrier is the desired conformation of the protein. Despite the challenges

of dividing the complex calculation over a network, Folding@Home has proven that using distributed computing is a viable solution to the protein folding computational and physical problem[9].

In October 2000, the project was launched. Since that time, more than 40,000 participants have actively contributed to the simulations, accumulating 10,000 CPU-years in approximately 12 months[9]. While this solution to the protein folding problem is helpful in reducing the time required for simulation, it remains computationally expensive and depends on computer architectures and networks.

It is clear that while progress has been made to reduce the time needed to simulate protein folding, the number of elements being computed is far to extensive and further reduction in computation is needed.

2.4.2 NAMD

NAMD is well known for its performance on large parallel computers but the program is actually used on many platforms, including laptops. This versatility is a great benefit for initiating and testing modeling projects. NAMD permits a novice to carry out standard simulations of most types readily, but NAMD also supports more advanced uses.

The purpose of NAMD is to enable high-performance MD simulation of molecules in realistic environments of 100,000 atoms or more. A decade ago in its first release,[10][11] NAMD permitted simulation of a protein-DNA complex encompassing 36,000 atoms[8] one of the largest simulations carried out at the time. The most recent release permitted the simulation of a protein-DNA complex of 314,000 atoms[16]. To probe the behavior of this 10-fold larger system, the simulated period actually increased 100-fold as well.

A common notation when discussing algorithm efficiency is by order. In this case, the algorithm for simulation is $O(n^2)$ (order of n^2), where n is the number of elements involved in the simulation. While NAMD simulation is promising, it is clear that the scaling of the algorithm is where the real problem lies. Meanwhile, the only reasonable conclusion is to reduce the elements involved in the simulation.

Further, the limits of NAMD's parallel scalability are mainly determined by these elements, in this case the atom count, with one processor per 1000 atoms being a conservative estimate for good efficiency on recent platforms. Once again, the limitation of a biological simulation relies upon the number of elements being processed.

Chapter 3

Methods

3.1 NAMD

3.1.1 Mechanics

NAMD performs atomic simulations where the atoms move according to the Newtonian equations of motion

$$m_\alpha \ddot{\vec{r}}_\alpha = -\frac{\delta}{\delta \vec{r}_\alpha} U_{total}(\vec{r}_1, \vec{r}_2, ..., \vec{r}_N), \alpha = 1, 2, ..., N$$

where m_α is the mass of the atom α, $\vec{r}_\alpha$ is its position, and U_{total} is the total potential energy that depends on all atomic positions and, thereby, couples the motion of the atoms. The potential energy can be represented as the MD force field and is the most crucial part of the simulation since it must represent the interaction between atoms[12].

The computational techniques only provide the ability to approximate these solutions. NAMD uses an all-atom MD simulation which assumes that every atom experiences a model force field which accounts for the interaction between an indi-

vidual atom and all the other atoms in the simulations.

The force field or the potential must because expressed as a summations

$$U_{total} = U_{bond} + U_{angle} + U_{dihedral} + U_{vdW} + U_{Coulomb}$$

where the first three terms of the total potential can be represented as individual summations

$$U_{bond} = \sum_{bonds\ i} k_i^{bond}(r_i - r_{0i})^2$$

$$U_{angle} = \sum_{angles\ i} k_i^{angle}(\theta_i - \theta_{0i})^2$$

$$U_{dihedral} = \sum_{dihedral\ i} \begin{cases} k_i^{dihe}[1 + cos(n_i\phi_i - \gamma_i)], & \text{if } n_i \neq 0 \\ k_i^{dihe}(0_i - \gamma_i)^2, & \text{if } n = 0 \end{cases}$$

and describe the stretching, bending, and torsional bonded interactions between atoms.

While this establishes a computational process for atomic simulation, we must address other factors that molecules experience in solution (i.e. temperature and pressure and pH).

To account for these factors, the Newtonian equations of motion are modified so that the computed short time step can still be interpretted correctly. For this, NAMD uses a stochastic coupling approach to enhance the dynamic stability of the amino acid[12].

The stochastic Langevin equation is used in NAMD to generate the Boltzmann distribution, a probability measure for the distribution of the states of a system, for canonical ensemble simulations. The generic Langevin equation is

$$M\dot{v} = F(r) - \gamma v + \sqrt{\frac{2\gamma k_B T}{M}} R(t)$$

where M is the mass, $v = \dot{r}$ is the velocity, F is the force, r is the position, γ is the friction coefficient, k_B is the Boltzmann constant, T is the temperature and $R(t)$ is a univariate Gaussian random process[12]. This equation ultimately governs the simulation interaction between the atoms in the simulated molecule.

3.1.2 Implementation

For the purpose of simulating small atom counts, NAMD performs extremely well and efficiently. What is apparent, though, is that larger molecules are difficult to simulate. NAMD's ease of use serves as the experimental mechanism for the data generated in this report.

First, a protein structure file (.psf) is generated for the amino acid from the Protein Data Bank file (.pdb) using the psfgen package made available in VMD[7]. While structures in the Protein Data Bank file hold the static information of the amino acid, the process of creating the structure file gives dynamic information which will be used in the simulation performed by NAMD. With these two files and using the simulation parameters shown in Table 3.1, simulations of a single amino acid may be performed. Such simulations allow observance of the atomic movement of the individual atoms in the amino acid through self-interactions.

With the execution of the simulation, the coordinate data is stored in binary trajectory files (.dcd) to be read and manipulated in the analysis portion of the experiment.

parameter	value	comments
structure	`${prefix}.psf`	protein structure file for the amino acid
coordinates	`${prefix}.pdb`	protein database file for the amino acid
parameters	`par_all27_prot_lipid.inp`	force-field parameters for proteins and amino acids
temperature	310	(K) human body temperature for the Langevin dynamics
timestep	2.0	(fs) time-step of simulation must be small
rigidBonds	all	needed for $2fs$ steps
langevin	on	do langevin dynamics
langevinDamping	1	(ps^{-1}) damping coefficient
restartfreq	500	every $1ps$
dcdfreq	500	data is captured every $1ps$

Table 3.1: NAMD Simulation Parameters

3.2 Molecular Dynamics Analysis

3.2.1 Amino Acid Backbone Stiffness Measurements

The python package MDAnalysis is employed in order to analyze the binary trajectory files created by NAMD. A coordinate representation of the molecule simulation data is then available in human-readable data form (.hrc). The MDAnalysis package facilitates the analysis of the results from NAMD. By treating the system as a single variable, all the information needed for molecular dynamics analysis is available.

Using the coordinates of the atoms, a list of structural angles for the amino acid is made. This list of the structural angles made by three atoms in the amino acid structure is then recorded. The angle is calculated by creating two vectors from the three atoms' coordinates that form the angle. After normalizing the vectors, the dot product between the two is taken and the arccosine yields the desired angles.

$$\theta = acos(v_1 \cdot v_2)$$

A histogram of these angles is created to observe the distribution of angles within the amino acid structure.

By understanding the dynamics of the amino acid's structural angles under self-interaction and thermal agitation, their conformational dynamics can be understood. If an angle remains fairly enclosed in a certain region of the histogram, then it can be treated as a fixed angle reducing number of computational objects required for the simulation.

3.2.2 Planar Dihedrals for Ramachandran Plotting

Each atom of the amino acid can be assigned to either the amide, carboxyl, or residue groups and a plane is defined to be the coordinates of three atoms that exist within their amino acid groups as shown in Figure 3.1.

To calculate the dihedral angle, two vectors made by the atoms which characterize the plane are used to find the normal vector by calculating the cross product between the two. This normal vector is unique to each plane. Now using the same calculation as was done in Section 3.2.1, the dihedral angle is calculated.

As was discussed in Section 2.3, the Ramachandran plots for the individual amino acids can be resolved from the simulation. For the dihedral angle, Ψ, the carboxyl and residue planes are used. For the dihedral angle, Φ, the amino and residue planes are used. In each case, the residue will be used as reference at each time step to factor out the orientation of the amino acid as is taken into account in stereoscopic experimentation described in Section 2.2.

These angles are then plotted against each other in a Ramachandran plot to validate the simulation against in vitro data.

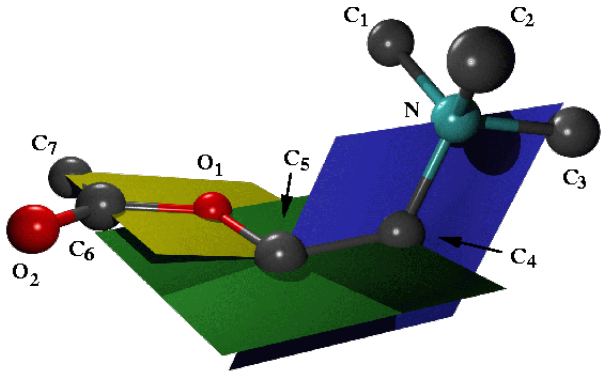


Figure 3.1: The carboxyl (yellow), amide (blue), and residue (green) groups of atoms within the amino acids can reduce their atomic representation to a planar one.

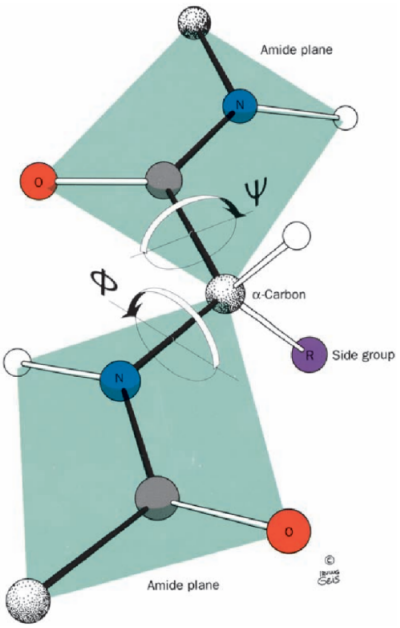


Figure 3.2: The Ψ and Φ dihedral angles in amino acids and the planes used to determine them.

Chapter 4

Results

4.1 Angle Measurements

The dynamics of detailed atomic models of biomolecules are traditionally limited to the nanosecond timescale[9]. NAMD demonstrates that traditional parallel molecular dynamics simulations using the numerical integration of Newton's equations can break the microsecond barrier. For these experiments, the timescale chosen is slightly short of the microsecond barrier to yield sufficient data for conclusions while remaining computationally inexpensive for the simulation.

Using the methods detailed in Section 3.2.1, a series of histograms are generated for the bond angles in the amino acid, glycine. Glycine (shown in Figure 4.1) is chosen because it is the smallest and most basic of the amino acids due to the residue only being a hydrogen atom. This choice allows for the most information to be gathered from the least amount of simulation since glycine only has 10 atoms.

The histograms are formed by 8×10^5 data points collected over 4×10^8 time steps during a simulation time of $8 \times 10^{-7} s$. By selecting the histograms of key groups, it can be seen in Figures 4.2 and 4.3 that those bond angles that exist within the amino

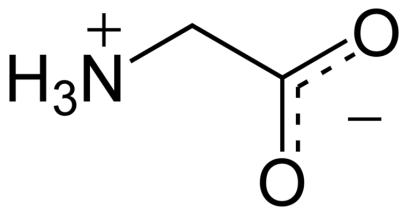


Figure 4.1: The chemical representation of the simplest amino acid, Glycine, is shown. The H_3N^+ group is known as the Amide group and the OOH group is known as the Carboxyl group. Here, the Residue group is not represented since it is a hydrogen atom.

and caboxyl planes have a well-defined and consistent sharp peak. It is distinct and while the bond can exist in other states, it is clear that more time is spent in a smaller angle range.

Conversely, by observing the bond angles in the adjoining bonds for the groups of the amino acid shown in Figure 4.4, no peak or preference of angle is observed. Instead, a gaussian-like distribution of angles can be seen. This demonstrates that there is less predictability to the position of the angles that connect the amino acid groups. The underlying dynamics of the amino acids motion clearly take place in the movement of these bond angles.

4.2 Ramachandran Reproduction

Histogram data for Ψ and Φ is used to generate a Ramachandran plot for the glycine amino acid. Previous experimentation exists which demonstrates a well known glycine mapping within these Ramachandran plots.

Figure 4.5 shows that despite the inability of the algorithm from Section 3.2.2 to produce the proper reflection, the same region of the map is occupied in the simulation results as was obtained in well-documented experimental results.

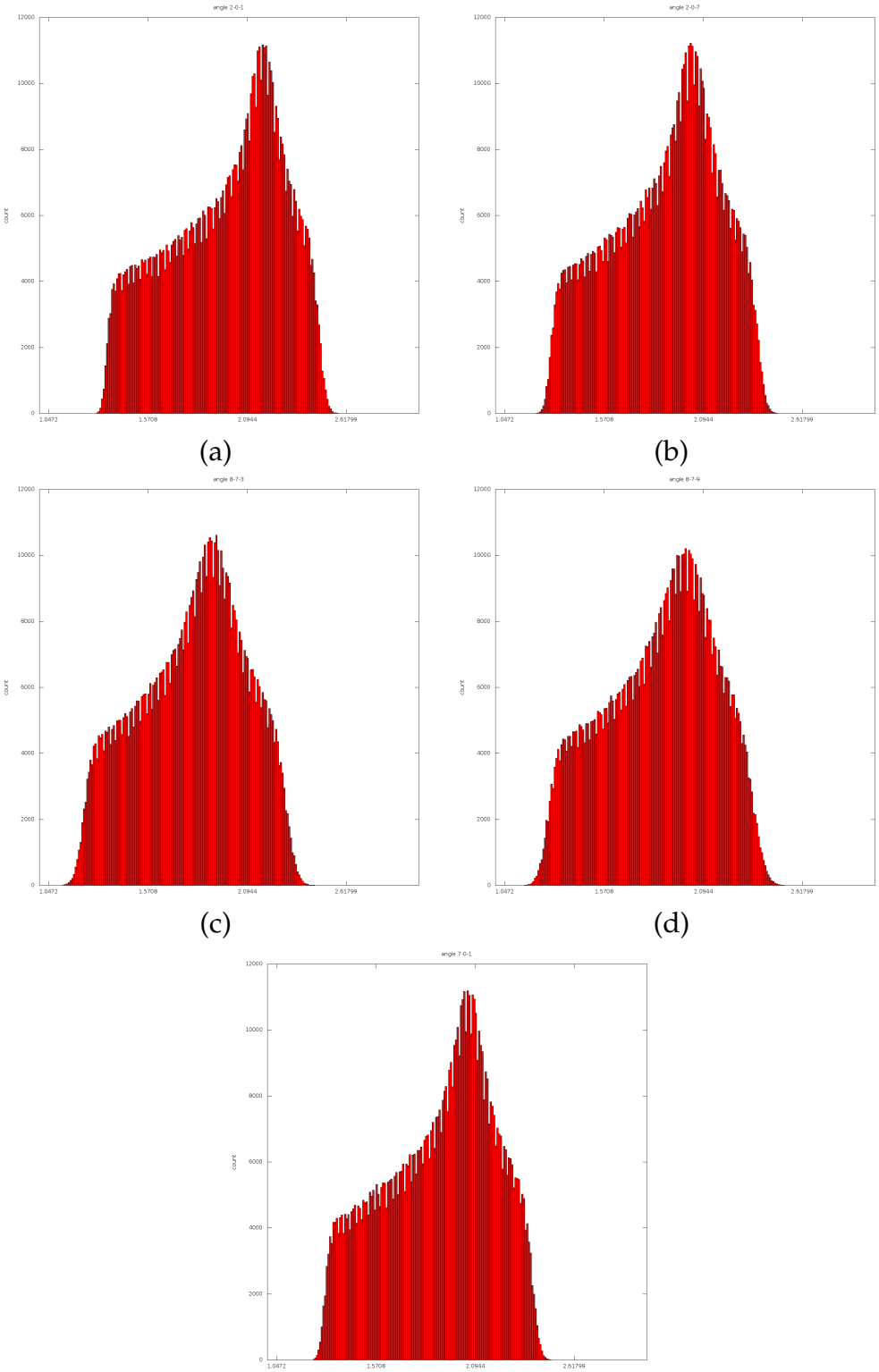
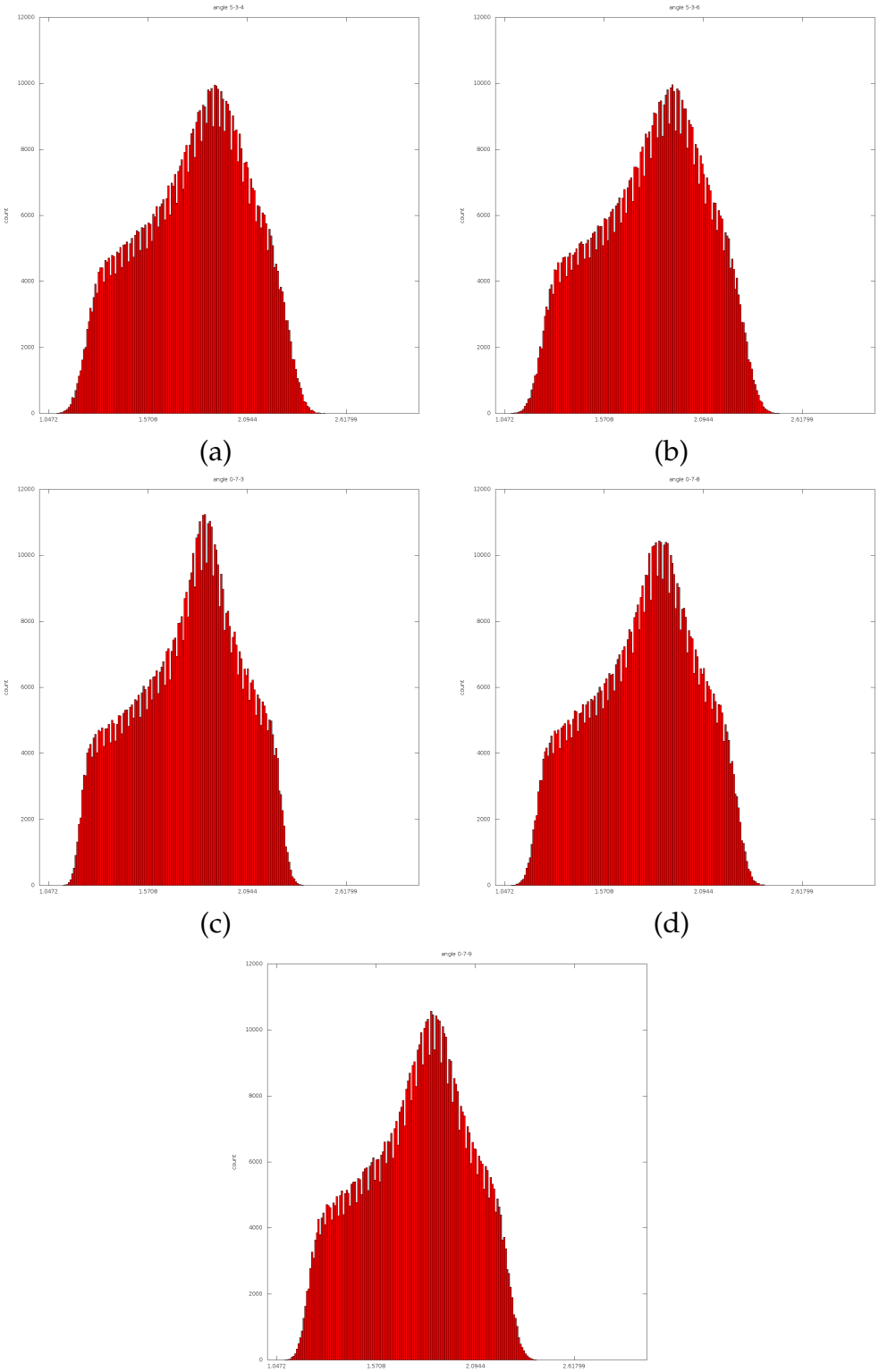
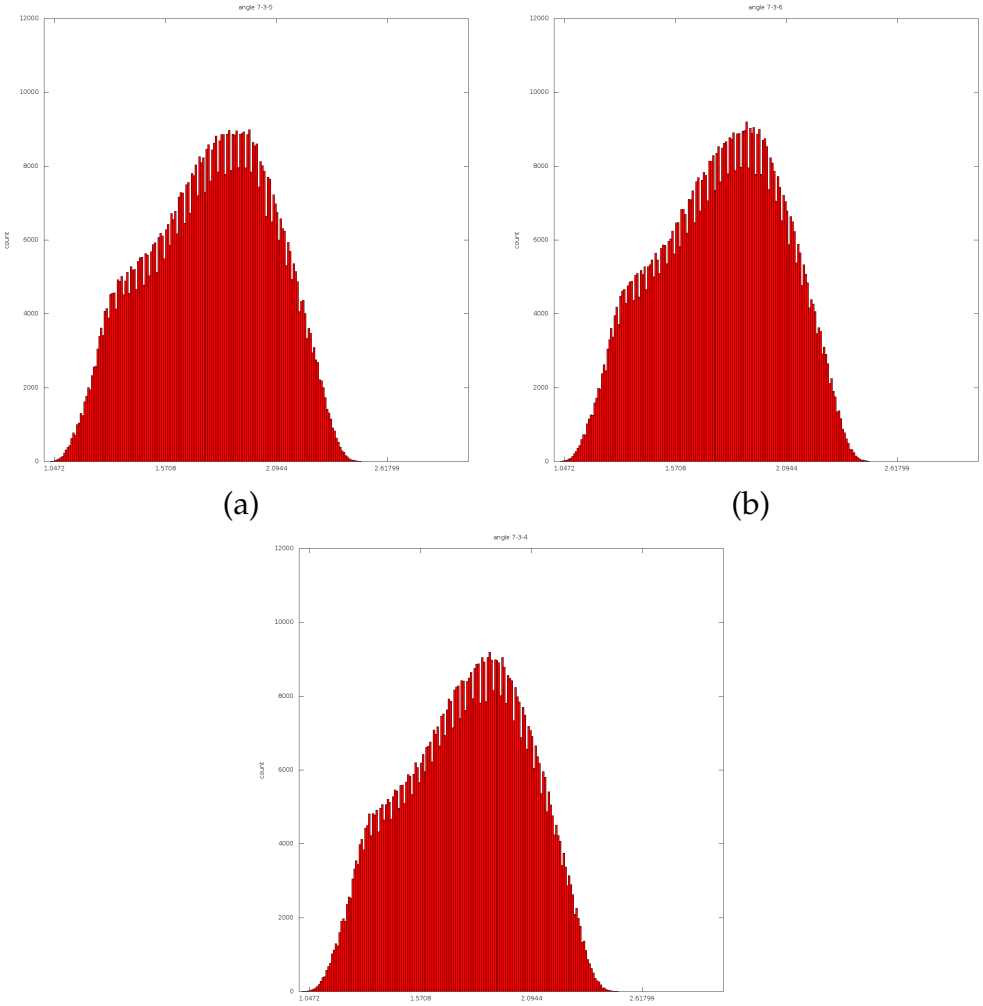


Figure 4.2: Histograms of Carboxyl Plane Angles:
(a) O-C-O (b) O-C-C (c) H-C-N (d) H-C-H (e) C-C-O



(a) (b) (c) (d) (e)

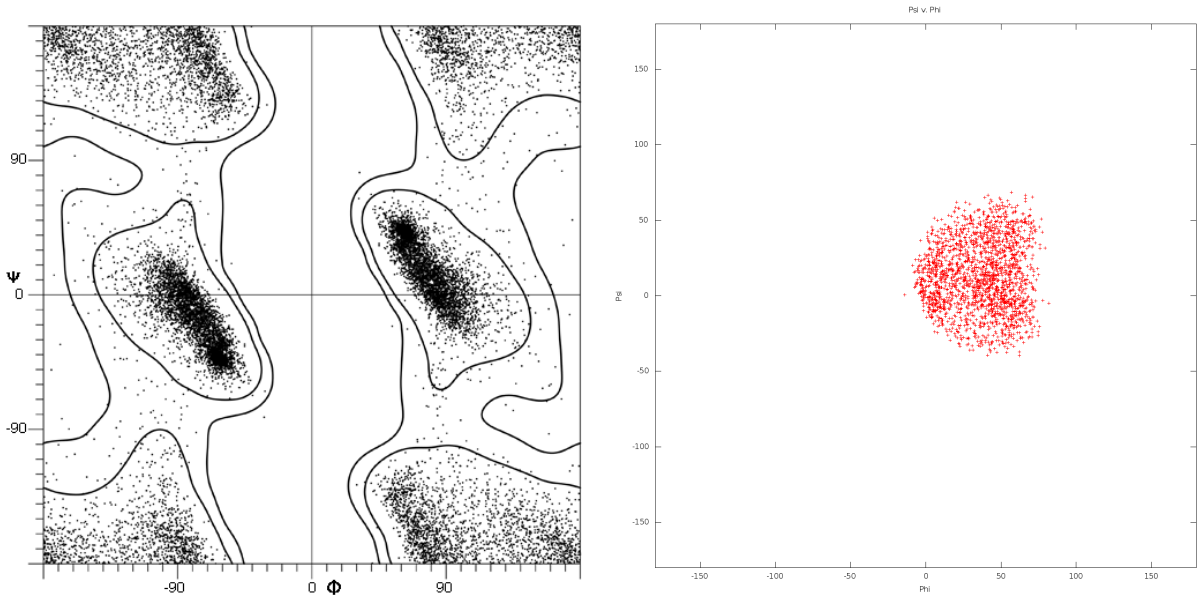
Figure 4.3: Histograms of Amide Plane Angles:
(a) H-N-H (b) H-N-H (c) C-C-N (d) C-C-H (e) C-C-H



(a) (b)

(c)

Figure 4.4: The bond angles that connect the (a) amide, (b) carboxyl, and (c) residue groups.



(a) (b)

Figure 4.5: Ramachandran plots show dihedral angles Ψ vs. Φ for glycine demonstrated in both (a) the literature and (b) simulation results.

Chapter 5

Discussion

5.1 Proof of Concept

As demonstrated in Section 3.2.1, there exist peaks in the histograms for the bonds which exist in the amide and carboxyl planes suggesting that there is a the bond primarily exists at and around that particular angle. For this reason, the assumption is made that the amide and carboxyl groups, as suggested in Section 3.2.2, may be treated as planes that rotate as described in Section 2.3. This means that the model for amino acid simulation may be constrained. Since the angles are constrained, we can treat the planes as objects instead of the atoms. This reduces the necessary simulation to three planes which rotate based on the dynamics given in the literature of the Ramachandran plots for singular amino acids. Using this further constraint, the simulation time of amino acids can be reduced by at least a factor of three (in the smallest amino acid, glycine) and at most by a factor of ten (in the largest amino acid, tryptophan). Though the algorithms have not been improved, the reduction in the amount of elements to be considered for computation has been demonstrated and is shown to be a plausible method for simulation.

5.2 Ramachandran Rout

The Ramachandran Plots that were simulated using the planar three-vector method described in Section 3.2.2 were partially recreated. What is readily apparent is that there is an issue in properly reflecting the data across the axis. This could be in part due to the geometrical calculations done or could be due to the orientation determining the angles. In either event, the algorithm is partially flawed and may be observed in Appendix B for further review.

Despite this partial failure, it is important that these Ramachandran plots can replicate the experimental results of biologists using Langevin dynamics on the reduced planar model of the amino acid.

5.3 Future Work

Most of the amyloid diseases are associated with old age, when there is likely to be an increased tendency for proteins to become misfolded or damaged, coupled with a decreased efficiency of the molecular chaperone and unfolded proteins responses[3]. It is therefore essential that there be development in understanding the misfolding and aggregation to find effective strategies for combating increasingly common and highly debilitating diseases[4].

Future work is being done at KBioSim to develop a linking algorithm to join the amino acids in their planar representation with a peptide bond formation. This will allow for the construction of polypeptide chains which will show whether or not the principle can apply to secondary and tertiary structures and result in a novel approach to protein folding.

Once this is achieved, it should be possible to use a similar Langevin dynamics

physics engine with a proprietary algorithm to further reduce calculations involving distance while folding.

Appendix A

Dependencies

A.1 NAMD

NAMD was developed by the Theoretical and Computational Biophysics Group in the Beckman Institute for Advanced Science and Technology at the University of Illinois at Urbana-Champaign. It is available for download at:
http://www.ks.uiuc.edu/Research/namd/

A.2 VMD

VMD was developed by Humphrey, W., Dalke, A. and Schulten, K. with funding from the National Institute of Health. It is available for download at:
http://www.ks.uiuc.edu/Research/vmd/

A.3 MDAnalysis

MDAnalysis is an open source python library available under the GNU GPL v2 code license. It is available for download at:

http://code.google.com/p/mdanalysis/

A.4 gnuplot

Gnuplot's source code is copyrighted but freely distributed. It is available for download at:

http://www.gnuplot.info/

Appendix B

Code

The working version detailed below is available for download at:

http://users.wpi.edu/~andershokinson/mqp.tar.gz

B.1 Simulation

B.1.1 namd.sh

```
for files in *.pdb; do

        steps="1000000";

        prefix="${files%.*}"

        echo    "package require psfgen
                        topology ../namd/top_all27_prot_lipid.inp
                        pdbalias residue HIS HSE
                        pdbalias atom ILE CD1 CD
                        segment U {pdb ${prefix}.pdb}
                        coordpdb ${prefix}.pdb U
                        guesscoord
                        writepdb ${prefix}.pdb
                        writepsf ${prefix}.psf

                        quit" >> ${prefix}.pgn

    echo "preparing ${prefix}.pdb for simulation"
        vmd -dispdev text -e ${prefix}.pgn >> ${prefix}.vmd
```

```
echo     "############################################################
         ## JOB DESCRIPTION                                        ##
         ############################################################

         # Minimization and Equilibration of
         # ${prefix} in a Water Sphere


         ############################################################
         ## ADJUSTABLE PARAMETERS                                  ##
         ############################################################

         structure          ${prefix}.psf
         coordinates        ${prefix}.pdb


         firsttimestep      0


         ############################################################
         ## SIMULATION PARAMETERS                                  ##
         ############################################################

         # Input
         paraTypeCharmm     on
         parameters         ../namd/par_all27_prot_lipid.inp
         temperature        310


         # Force-Field Parameters
         exclude            scaled1-4
         1-4scaling         1.0
         cutoff             12.0
         switching          on
         switchdist         10.0
         pairlistdist       14.0


         # Integrator Parameters
         timestep           2.0  ;# 2fs/step
         rigidBonds         all  ;# needed for 2fs steps
         nonbondedFreq      1
         fullElectFrequency 2
         stepspercycle      10


         # Constant Temperature Control
         langevin           on    ;# do langevin dynamics
         langevinDamping    1     ;# damping coefficient (gamma) of 1/ps
         langevinTemp       310
         langevinHydrogen   off    ;# don't couple langevin bath to hydrogens
```

```
# Output
outputName          ${prefix}

restartfreq         500     ;# 500steps = every 1ps
dcdfreq             500
outputEnergies      100
outputPressure      100


#############################################################
## EXTRA PARAMETERS                                        ##
#############################################################

#############################################################
## EXECUTION SCRIPT                                        ##
#############################################################

# Minimization
minimize            100
reinitvels          310

run ${steps} ;# 5ps" >> ${prefix}.conf

echo "performing simulation using ${prefix}.psf and namd"
namd2 +idlepoll ${prefix}.conf >> ${prefix}.namd

rm *.xsc *.coor *.conf *.pgn *.old *.vel *.vmd *.namd

done
```

B.2 Analysis

B.2.1 analysis.sh

```
# cp analysis/dihedral/data.py .
cp analysis/angle/data.py

for files in *.dcd; do
        prefix="${files%.*}"

        python -c "import data; data.angles('${prefix}')"
        # python -c "import data; data.dihedrals('${prefix}')"

done

rm data*
```

B.2.2 data.py

```
# Anders Hokinson
# 2012

from MDAnalysis import * # only allowed at module level

def get_angle(coordinate_1, coordinate_2, coordinate_3):
        import math

        v1 = [coordinate_1[0] - coordinate_2[0], \
                  coordinate_1[1] - coordinate_2[1], \
                  coordinate_1[2] - coordinate_2[2]]

        v1mag = math.sqrt(v1[0] ** 2 + \
                                        v1[1] ** 2 + \
                                        v1[2] ** 2)

        v1n = [ 0, 0, 0]

        for i in range(len(v1) - 1):

                v1n[i] = v1[i] / v1mag

        v2 = [coordinate_3[0] - coordinate_2[0], \
                  coordinate_3[1] - coordinate_2[1], \
                  coordinate_3[2] - coordinate_2[2]]

        v2mag = math.sqrt(v2[0] ** 2 + \
                                        v2[1] ** 2 + \
                                        v2[2] ** 2)

        v2n = [ 0, 0, 0]
        for i in range(len(v2) - 1):
                v2n[i] = v2[i] / v2mag

        dot = v1n[0] * v2n[0] + \
                  v1n[1] * v2n[1] + \
                  v1n[2] * v2n[2]

        angle = math.acos(dot) / (2*math.pi) * 360
        return angle

def angles(prefix):
        psf = str(prefix)
        psf = psf[:len(psf)-1] + '.psf'
        dcd = str(prefix) + '.dcd'
        hrc = str(prefix) + '.hrc'
        print 'using files %s and %s to create %s' % (psf, dcd, hrc)

        universe = Universe(psf, dcd)
```

```
f = open(hrc, 'w')
print 'opening %s for writing' % (hrc)
bonds = []

coordinates = universe.atoms.coordinates()

print 'calculating number of angles in %s' % (psf)
for atom in universe.atoms.indices():

        vertex = coordinates[atom]

        for bond in universe.bonds:

                if bond[0] == atom:
                        bonds += [bond[1]]
                if bond[1] == atom:
                        bonds += [bond[0]]
        if len(bonds) > 1:
                for i in bonds:
                        for j in bonds:
                                if i != j:
                                        anglename = str(i)+'-'+str(atom)+'-'+str(j)
                                        f.write(anglename+'\t')

                        bonds.pop(0)

        bonds = []
f.write('\n')


print 'writing angles to %s' % (hrc)
for ts in universe.trajectory:
        coordinates = universe.atoms.coordinates()

        for atom in universe.atoms.indices():
                vertex = coordinates[atom]

                for bond in universe.bonds:

                        if bond[0] == atom:
                                bonds += [bond[1]]
                        if bond[1] == atom:
                                bonds += [bond[0]]

                if len(bonds) > 1:
                        for i in bonds:
                                for j in bonds:
                                        if i != j:
                                                angle = get_angle(coordinates[i], vertex, coordinates[j])
                                                writeangle = '%.3f' % (angle)
                                                f.write(writeangle+'\t')
```

```
                                        bonds.pop(0)

                        bonds = []
                f.write('\n')

        f.close()
        print 'completed successfully, check %s for results' % (hrc)
```

B.2.3 data.py

```
# Anders Hokinson
# 2012

from MDAnalysis import * # only allowed at module level

def get_dihedral(plane, residue):
        import math

        plane1=plane[0]
        plane2=plane[1]
        plane3=plane[2]

        residue1=residue[0]
        residue2=residue[1]
        residue3=residue[2]

        v1p = [plane1[0]-plane2[0], \
               plane1[1]-plane2[1], \
               plane1[2]-plane2[2]]

        v2p = [plane3[0]-plane2[0], \
               plane3[1]-plane2[1], \
               plane3[2]-plane2[2]]

        np = [v1p[1]*v2p[2]-v1p[2]*v2p[1], \
              v1p[2]*v2p[0]-v1p[0]*v2p[2], \
              v1p[0]*v2p[1]-v1p[1]*v2p[0]]

        npmag = math.sqrt(np[0] ** 2 + \
                                        np[1] ** 2 + \
                                        np[2] ** 2)

        nprho = npmag
        npphi = math.atan2(np[1]/np[0])
        nptheta = math.acos(np[2]/nprho)

        npn = [0]*3
        for i in range(len(np) - 1):
                npn[i] = np[i] / npmag

        r=[residue1[0]-residue2[0], \
```

```
            residue1[1]-residue2[1], \
            residue1[2]-residue2[2]]

        rmag=math.sqrt(r[0] ** 2 + \
                                r[1] ** 2 + \
                                r[2] ** 2)

        rrho = rmag
        rphi = math.atan2(r[1]/r[0])
        rtheta = math.acos(r[2]/rrho)

        rn = [0]*3
        for i in range(len(r) - 1):
                rn[i] = r[i] / rmag

        dot = npn[0] * rn[0] + \
                  npn[1] * rn[1] + \
                  npn[2] * rn[2]

        asinv = math.asin(dot) / (2*math.pi) * 360
        acosv = math.acos(dot) / (2*math.pi) * 360

        if asinv > 0:
                if acosv > 0:
                        dihedral = 90 - acosv
                else:
                        dihedral = 90 + acosv
        else:
                if acosv > 0:
                        dihedral = 90 - acosv
                else:
                        dihedral = -270 + acosv

        return dihedral


def dihedrals(prefix):
        psf = str(prefix) + '.psf'
        dcd = str(prefix) + '.dcd'
        kpp = str(prefix) + '.kpp'
        print 'using files %s and %s to create %s' % (psf, dcd, kpp)

        universe = Universe(psf, dcd)

        f = open(kpp, 'w')

        coordinates = universe.atoms.coordinates()

        for ts in universe.trajectory:
                coordinates = universe.atoms.coordinates()

                carboxyl = [coordinates[0],coordinates[1],coordinates[2]]
```

```
            amino = [coordinates[3],coordinates[4],coordinates[5]]
            residue = [coordinates[7],coordinates[8],coordinates[9]]

            psi = get_dihedral(carboxyl,residue)
            phi = get_dihedral(amino,residue)

            writedihedral = '%.3f' % (psi)
            f.write(writedihedral+'\t')
            writedihedral = '%.3f' % (phi)
            f.write(writedihedral+'\t')

            f.write('\n')

        f.close()
```

B.3 Results

B.3.1 histogram.sh

```
for files in *.psf; do
        prefix="${files%.*}"
        mkdir ${prefix}

        bonds=`awk '{print NF}' ${prefix}1.hrc | sort -nu | tail -n 1`

        for ((i=1; i<$bonds+1; i++)); do

            bond=`awk 'NR==1 {print $'$i'}' ${prefix}1.hrc`
            echo    "processing angle $bond data"
            echo    "creating $bond.png"

            echo    "clear
                        reset

                        numbins = 360
                        binwidth = 1

                        set key off
                        set auto
                        set xrange[0:360]
                        set xtics 30
                        set title 'angle ${bond}'
                        set ylabel 'count'

                        set term png size 1280, 1280
                        set output \"${prefix}/${bond}.png\"

                        set style histogram clustered gap 1
                        set style fill solid border -1
```

```
                        set boxwidth binwidth

                        bin(x,width) = width*floor(x/width) + binwidth/2.0

                        plot '${prefix}${j}.hrc' every ::1 using (bin(\$${i},binwidth)):(1) smooth freq with boxes" >> angles.gp
                done

                echo 'load "angles.gp"' | gnuplot
                echo 'exit' | gnuplot
                rm angles.gp

        done

        mv ${prefix}* ${prefix}/

done
```

B.3.2 psiVphi.sh

```
date="$(date +%m%d%H%M)"
mkdir results/${date}/

for files in *.psf; do
        prefix="${files%.*}"

        mkdir results/${date}/${prefix}


        echo    "creating ${prefix}.png"

        echo    "clear
                        reset

                        set key off
                        set yrange[-180:180]
                        set xrange[-180:180]

                        set title 'Psi v. Phi'
                        set ylabel 'Psi'
                        set xlabel 'Phi'

                        set term png size 1280, 1280
                        set output \"${prefix}.png\"



                        plot '${prefix}.kpp' " >> angles.gp

        echo 'load "angles.gp"' | gnuplot
        echo 'exit' | gnuplot
        rm angles.gp
```

```
        mv ${prefix}* results/${date}/${prefix}

done
```

B.4 Scripting

B.4.1 load.sh

```
#!/bin/bash

load() {

for i in $@; do
        cp ../pdb/${i}* ./working
done
}

load 'gly'
```

B.4.2 kinn.sh

```
#!/bin/bash

cp namd/namd.sh working
./working/namd.sh
rm working/namd.sh

cp analysis/analysis.sh
./working/analysis.sh
rm working/analysis.sh

# cp plot/dihedral/psiVphi.sh working
cp plot/angle/histogram.sh working
# ./working/psiVphi.sh
./working/histogram.sh
# rm working/psiVphi.sh
rm working/histogram.sh
```

Bibliography

[1] Thomas F. Anderson. Stereoscopic studies of cells and viruses in the electron microscope, 1952.

[2] F. C. Bernstein. The protein data bank: A computer-based archival file for macromolecular structures, 1977.

[3] P. Csermely. Chaperone overload is a possible contributor to civilization diseases, 2001.

[4] C. M. Dobson. Protein folding and disease: a view from the first horizon symposium, 2003.

[5] C. M. Dobson, A. Sali, and M. Karplus. Protein folding: a perspective from theory and experiment, 1998.

[6] Christopher M. Dobson. Protein folding and misfolding, 2003.

[7] Tim Isgro, James Phillips, Marcos Sotomayor, Elizabeth Villa, Hang Yu, David Tanner, and Yanxin Liu. Namd tutorial: Unix/macosx version, February 2012.

[8] D. Kosztin, T. C. Bishop, and K. Schulten, 1997.

[9] Stefan M. Larson, Christopher D. Snow, Michael Shirts, and Vijay S. Pande.

Folding@home and genome@home: Using distributed computing to tackle previously intractable problems in computational biology.

[10] M. Nelson, W. Humphrey, A. Gursoy, A. Dalke, L. Kale, R. Skeel, and K. Schulten, 1996.

[11] M. Nelson, W. Humphrey, A. Gursoy, A. Dalke, L. Kale, R. Skeel, K. Schulten, and R. Kufrin, 1995.

[12] James C. Phillips, Rosemary Braun, Wei Wang, James Gumbart, Emad Tajkhorshid, Elizabeth Villa, Christophe Chipot, Robert D. Skeel, Laxmikant Kal, and Klaus Schulten. Scalable molecular dynamics with namd, 26 May 2005.

[13] G. N. Ramachandran, C. Ramakrishnan, and V. Sasisekharan. Stereochemistry of polypeptide chain configurations, 1963.

[14] B. Schuler, E. A. Lipman, and W. A. Eaton. Probing the free-energy surface for protein folding with single-molecule fluorescence spectroscopy, 2002.

[15] Zhengshuang Shi and Neville R. Kallenbach. Ramachadran redux. *PNAS*, 108(1), 2011.

[16] E. Villa, A. Balaeff, and K. Schulten, 2005.