DEVELOPMENT OF A MINIMAL POLYMER MODEL FOR THE DESCRIPTION
OF β-HAIRPIN FORMATION

A Thesis
Presented to
The Graduate Faculty of The University of Akron

In Partial Fulfillment
of the Requirements for the Degree
Master of Science

Kenneth E. Milam
August 2006
DEVELOPMENT OF A MINIMAL POLYMER MODEL FOR THE DESCRIPTION
OF β-HAIRPIN FORMATION

Kenneth E. Milam

Thesis

Approved: ___________________________ Accepted: ___________________________

Advisor __________________________________________________________
Dr. Gustavo Carri

Dean of the College ________________________________________________
Dr. Frank N. Kelley

Faculty Reader _____________________________________________________
Dr. Ali Dhinojwala

Dean of the Graduate School __________________________________________
Dr. George R. Newkome

Department Chair __________________________________________________
Dr. Mark Foster

Date
The purpose of this thesis is to develop a coarse-grained, three dimensional, free-space model of a polypeptide capable of folding into a beta-sheet conformation. The model is built on a modified Freely Jointed Chain (FJC) model of polymers where each residue is modeled as a hard sphere. For simplicity, all residues are considered to be chemically identical. The equilibrium value of the bond angles is set to 120 degrees. If any bond angle deviates from this value then an energetic penalty is applied. In addition, an energetic contribution is applied whenever several geometrical conditions, used to determine the formation of hydrogen bonds, are satisfied. The latter term is negative which stabilizes, in a thermodynamic sense, the formation of hydrogen bonds at low temperatures. All these concepts were employed in a Monte Carlo simulation that we used to solve the model.

The first result of our study was the density of states of the system which we used to compute the thermodynamic properties of the model. The most important finding of this study was a very rich phase diagram for this relatively simple system. Indeed, the phase diagram showed clear transitions among many states that need further investigation. However, we were able to identify some of these states. For example, different regions of the phase diagram favored different conformations of the chain.
Three of these conformations were identified: the extended one, i.e. rod-like, the beta hairpin, and the random coil. We discuss these and other results hereafter.
ACKNOWLEDGEMENTS

There are several people that helped make this work possible. First of all is my advisor, Dr. Carri. Without the opportunities and guidance he gave me, as well as his motivation, I would not have been able to achieve this work. I would also like to thank Dr. Dhinojwala for serving as the reader for this thesis and the Department of Polymer Science for supporting me for the past two years as well as giving me an unparalleled education.

I would also like to thank all the members of the group. In particular, I thank Vikas Varshney for his extensive help during the development of this thesis. I also appreciate the help given by all of my other group members in troubleshooting this work; Taner Dirama, Pavan Ghatty and Andrew Nathan.

My family has always been a strong motivational force for me. Without their support and confidence in my abilities I would not have been able to stay the course.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>LIST OF TABLES</th>
<th>viii</th>
</tr>
</thead>
<tbody>
<tr>
<td>LIST OF FIGURES</td>
<td>ix</td>
</tr>
</tbody>
</table>

**CHAPTER**

**I. INTRODUCTION**

1.1. Proteins

1.2. Description of the Problem

1.3. Literature Review

1.3.1. Experimental Work

1.3.2. Molecular Dynamics

1.3.3. Monte Carlo

1.4. Monte Carlo Computer Simulations

**II. THE MODEL**

2.1 Physical Foundations

2.2 Chain Generation Algorithm

2.3 Energy Function

2.4 Kink Algorithm

---

vi
2.5 Pivot Algorithm ................................................................. 37
2.6 Histogram .................................................................. 37
2.7 Density of States ............................................................ 37
2.8 Program Utilization ......................................................... 38
III. ANALYSIS ........................................................................... 41
  3.1 Preliminary Data ............................................................. 41
  3.2 Data Analysis Program ..................................................... 43
  3.3 Phase Diagram Program .................................................. 45
IV. CONCLUSIONS ................................................................. 64
V. FUTURE WORK ................................................................. 65
REFERENCES ............................................................................. 67
APPENDICES .............................................................................. 70
  APPENDIX A. TABLE OF α-AMINO ACIDS FOUND IN PROTEINS ...... 71
  APPENDIX B. MAIN PROGRAM ............................................... 74
  APPENDIX C. DATA ANALYSIS PROGRAM ................................. 111
  APPENDIX D. PHASE DIAGRAM PROGRAM ............................... 119
# LIST OF TABLES

<table>
<thead>
<tr>
<th>Table</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>40</td>
</tr>
</tbody>
</table>

Feasible Bins
# LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1 Parallel Beta Strands</td>
<td>26</td>
</tr>
<tr>
<td>1.2 Anti-parallel Beta Strands</td>
<td>27</td>
</tr>
<tr>
<td>1.3.1 The NOE’s for the backbone CαH—NH are represented by the solid arrows. These indicate a parallel orientation of the beta strands</td>
<td>28</td>
</tr>
<tr>
<td>1.3.2 Probability that the peptide chain will be in any given conformation based on the number of hydrogen bonds and the radius of gyration</td>
<td>29</td>
</tr>
<tr>
<td>1.3.3 Monomer A (Filled) is used to stabilize the strands in specific configurations by preferably bonding with other A monomers and dictating the location of the turns</td>
<td>30</td>
</tr>
<tr>
<td>1.3.4 Histogram Distribution of First Folding Time</td>
<td>31</td>
</tr>
<tr>
<td>1.4.1 Energy landscape example</td>
<td>32</td>
</tr>
<tr>
<td>3.1. Density of States:</td>
<td></td>
</tr>
<tr>
<td>a) Top View of the 10-bead system</td>
<td>47</td>
</tr>
<tr>
<td>b) Side View of the 10-bead system</td>
<td>48</td>
</tr>
<tr>
<td>3.2 Partition Function of the 10-bead system</td>
<td>49</td>
</tr>
<tr>
<td>3.3 Free Energy of the 10-bead system</td>
<td>50</td>
</tr>
<tr>
<td>3.4 Internal Energy of the 10-bead system</td>
<td>51</td>
</tr>
<tr>
<td>3.5 Entropy of the 10-bead system</td>
<td>52</td>
</tr>
</tbody>
</table>
3.6 Heat Capacity of the 10-bead system
   a) Angle View .................................................................53
   b) Side View........................................................................54

3.7 Phase Diagram for the 10-bead system with a C parameter of -650...............55

3.8 A Random Coil conformation of the 10-bead system ..............................56

3.9 An Extended Chain Conformation of the 10-bead system ......................57

3.10 A Beta Hairpin of the 10-bead system .............................................58

3.11 Phase Diagram for a 10-bead system with a C parameter of -275...............59

3.12 Density of States of the 16-bead system..........................................60

3.13 Entropy of the 16-bead system..................................................61

3.14 Heat Capacity of the 16-bead system............................................62

3.15 Phase Diagram for a 16-bead system with a C parameter of -650..........63
CHAPTER I

INTRODUCTION

1.1 Proteins

Since the discovery of the intimate relationship between protein structure and activity, the field of proteomics has grown at an accelerated rate.\textsuperscript{1,2,3,4} Indeed, protein structure, as determined by the primary, secondary, and tertiary structure, determines the dynamics of the protein which, in turn, control its activity. Thus, the ultimate goal of proteomics is to be able to predict how a protein will fold and function based on its amino acid sequence.\textsuperscript{1-5} Once this knowledge is gained, we will be able to manufacture proteins with very specific functions from super specific drugs to gene therapy.\textsuperscript{5,6}

Proteins are relatively complex macromolecules when compared to the typical synthetic polymer. Unlike conventional polymers, the repeat unit can be chosen from a pool of twenty different amino acids or residues. These residues, connected in a specific sequence, allow the protein to be chemically active so that it can perform its function. Amino acids have several common elements in their structure; one side has an amine group while the other side has a carboxyl group and, in the middle, there is a central carbon atom, known as the alpha carbon, \(\text{C}_\alpha\). However, the differences among the twenty different amino acids resides reside in the so-called “R” group, also known as the side chain for \(\text{C}_\alpha\). The chemical properties of the side chain range from polar to hydrophobic,
from acidic to basic and can even allow for disulfide bonds or hydrogen bonds between residues.

From a physics perspective, the biggest difference between the typical synthetic polymer and a protein is that the polymer folds into a random configuration that depends solely on the solvent and temperature, whereas a protein folds into the same conformation repeatedly even after being denatured. This conformation is called the native or folded state of the protein and is specified by the amino acid sequence in the protein, i.e. primary structure. The native state is a compact and chemically active conformation of the protein that is used to perform a function.

Proteins are designed to adopt specific shapes so that they can perform their functions. If the active sites of the protein are mis-folded or are not in the correct position, the protein cannot perform its function. This requires a protein to fold in a specific way each time it is synthesized or after being denatured.

A protein has many possible conformations including highly ordered structures. The most thermodynamically stable conformation is the native state which has the minimum free energy. Typical conformations have the residues with hydrophobic side groups folded inward and the polar residues on the outer shell of the protein due to the polar nature of water, the solvent inside biological systems. Actually, these hydrophobic side groups are known to play a very important role in governing the dynamics of a protein folding to its native state and have been discussed by others previously.

The structure of proteins is hierarchical by nature. The primitive structure of an individual protein is the amino acid sequence, also known as primary structure. There are two basic stabilized, or unstrained, secondary structures in folded proteins, the alpha
helix and the beta strand. The alpha helix conformation consists of several intra-protein hydrogen bonded amino acids that form a helix-shaped structure. A beta sheet is similar to an alpha helix in the sense that it is stabilized by intra-protein hydrogen bonds, but it differs in shape. Beta-sheets consist of parallel or anti-parallel residue strands, called beta strands, which are held together by hydrogen bonding interactions. The strands are considered to be parallel if they are oriented in the same direction (as shown in Fig. 1.17), and are anti-parallel if they are oriented opposite of each other (Fig. 1.27). In this thesis we are interested in modeling the beta-sheet structure.

A parallel beta sheet has relatively evenly spaced hydrogen bonds while the anti-parallel version has hydrogen bonds that alternate between being near to one another and being farther away from each other. The most common form of beta sheet is known as the beta hairpin, as it is shaped like a hairpin. It consists of two strands of amino acids hydrogen bonded together and a turn or loop of randomly oriented amino acids that connect the two strands. The turn in a beta hairpin structure can be a couple of residues long or hundreds, connecting opposite ends of the polypeptide chain, but they are typically from two to five residues in length. Loops can be chemically inactive, but often contain the reactive residues that allow the protein to function. These loops are normally located on the outside of the polymer chain, allowing the reactive residues to be able to interact with other molecules.6 The strands of amino acids forming the beta hairpin range from five to ten residues in length, and are oriented so that the carboxyl group of one residue can hydrogen bond with the amine group of another residue and vice versa. Hairpins normally contain anti-parallel beta strands. The next level up in complexity would be the beta sheet and it occurs when several residue strands hydrogen bond
together, or when one chain folds back on itself many times forming a sheet, similar to
the lamellar structure routinely observed in semicrystalline polymers. Sheets can be either
parallel or anti-parallel depending on the specific amino acids used. Protein strands can
also fold into beta barrel structures which are several beta-strands bonded together so that
the last beta-strand is bonded to the first one forming a round cylindrical “barrel” shaped
conformation which, like beta sheets, can contain either parallel or anti-parallel strands.
All these conformations occur in nature readily, but the processes that determine how
these conformations are formed are not fully understood, yet.²

Tertiary structures are next level in the hierarchal structure of proteins and are
combinations of secondary structures coupled together in the native state. Some examples
of tertiary structures would be bundles of alpha helixes or alpha-beta structures. Finally,
quaternary structures are actually several different molecules arranged together into one
complex conformation.

Stabilization of the native conformation of a protein is required to ensure that it
maintains its vital arrangement long enough to be utilized. Hydrogen bonding has been
known to be one of the main forces stabilizing the proteins native conformation as well as
a guide to folding.⁴ Mirsky and Pauly theorized this even before secondary structures in
proteins were discovered. They used solvents that affected the ability of a molecule to
hydrogen bond and obtained denatured proteins, and then allowed the proteins to refold.
As it refolds, a protein chain forms hydrogen bonds. These bonds help to stabilize the
structure by lowering the overall free energy as well as constraining the actual physical
motion of the residues. Amino acids must be in the correct geometric position to form
hydrogen bonds. The position of the residue needs to fulfill several conditions, the first of
which consists of the relative distance between the amino acids. The relative orientation then becomes a factor. Each amino acid has two points for hydrogen bonding, one amine hydrogen and one carbonyl oxygen. The corresponding parts of the amino acids must be oriented toward one another, i.e. the amine should be oriented toward the carboxyl group of the other residue. The angle at which the N-H bond faces the C=O bond must be within a specific angle to ensure a hydrogen bond.

Anfinsen and co-workers showed that not even a completely denatured protein can be prevented from folding back into its native state when the right experimental conditions are set. For this work they were awarded the Nobel Prize in Chemistry in 1972, specifically for using chemical analysis to prove that a “denatured” protein folds back into its native shape. This is one of the most important findings in the field of protein folding. Because proteins fold and refold into the same conformation, one should be able to predict the final native state with only the amino acid sequence. Hence, Anfinsen and co-workers proposed the “thermodynamic hypothesis” which states that the three-dimensional native structure of a protein must be the conformation with the lowest possible Gibbs free energy. This reinforces the concept that the three-dimensional structure of the protein is based on the specific amino acid sequence because the Gibbs free energy is a direct consequence of the inter-atomic interactions of the protein chain and its entropy. Using this hypothesis, models can be formulated for the simulation of protein conformations and the three-dimensional native structure could, in principle, be determined from the amino acid sequence directly; thus, comparison of the three-dimensional structure with experimental data would allow us to verify the previously
mentioned hypothesis. Unfortunately, this has yet to be achieved; however, several advancements have been made in pursuit of this endeavor in recent years.⁵,⁸,⁹,¹⁰

One of the biggest mysteries in the field of protein folding is the Levinthal paradox. Levinthal proposed that all conformations of a protein are equally probable;⁵,¹¹ therefore, the only way to achieve the native folded state is by randomly searching each individual conformation until the correct one is found. The statistical probability of a protein folding into its native state is relatively negligible. Even for small chains, the time it would take to randomly find the correct conformation of the protein is many orders of magnitude larger than the actual folding time. However, since proteins do fold into their native states within a fairly short time period, Levinthal states that no randomly guided folding process could uniformly reach the native state in such a short period of time.¹¹

The only way for a protein to fold into a specific conformation in a relatively short period of time requires the folding process to be directed along a certain pathway towards the native state. This mechanism which allows a protein to find its native state in seconds or milliseconds is now thought to be relatively understood.⁵,⁸,⁹,¹⁰,¹² The folding is “bias” toward the native state, in a so-called folding funnel which eliminates any irrelevant conformations.¹³ The folding funnel is large at the unfolded or denatured state, and narrows as the protein approaches its native state. The funnel is based on the free energy landscape of the different protein conformations. As the protein folds the hydrogen bonds and the hydrophobic side groups guide the protein by giving it benefits in terms of free energy. As the protein collapses it has fewer and fewer possible conformations, and its more compact form causes secondary structures to be necessary in order to avoid steric hindrance.
Recently, it has been shown that proteins have relatively little restriction to fold into a native state.\textsuperscript{14} The transition barriers from the random unfolded state to any intermediate states that might exist to the native state are much lower and broader than previously believed, allowing for an easier transition. This can be justified by the fact that transitioning from one individual state to another requires relatively small changes in free energies. The low broad transition barriers have also been demonstrated experimentally, showing that by changing the chain you can move, broaden, or even change the height of the peak. This also supports the folding funnel method for a protein to quickly reach its native state, if the transition barriers are less restrictive then the protein will continually move closer towards its energetic minimum.

Another proposed reason for proteins taking little time to fold is the two-phase folding. A protein will start out in an open, unfolded state. Initially the protein rapidly collapses to a set of relatively compact metastable states.\textsuperscript{12,15,16,17} The hydrophobic side groups of the protein seem to dominate this initial step, guiding themselves inward towards each other and away from the solvent. Then, the protein slowly reconfigures into additional metastable states, forming hydrogen bonds as it folds, until it reaches the native confirmation. Rapid folding is also influenced by a pronounced energy gap between the native state and most other configurations, meaning that the chain will continually reconfigure until reaching this minimum.\textsuperscript{1,18} High folding temperatures relative to the glass transition temperature causes the chain to fold more quickly as well, and the collapse temperature and the folding temperature should be relatively close. These allow for the protein to be able to move freely, if these conditions are not met the protein basically freezes in place.
The main focus of this thesis is to study the thermodynamics of the formation of beta hairpins from the unfolded state. Our work is based on the previous study of Dimitreiski and co-workers.\textsuperscript{18} In this thesis; I have attempted to further the field by analyzing the density of states given by a Monte Carlo simulation method of a coarse-grained model appropriate for the description of beta hairpins.

1.2 Description of Problem

The purpose of this work was to develop a feasible, coarse-grained, three dimensional, free space model of a homopolypeptide capable of folding into a beta hairpin structure. The model uses a modified Freely Jointed Chain (FJC) model of polymers where each residue is modeled as a hard sphere. The equilibrium value of the bond angles is set to 120 degrees. If any bond angle deviates from this value then an energetic penalty is applied. In addition, an energetic contribution is applied whenever several geometrical conditions, used to determine the formation of hydrogen bonds, are satisfied. The latter term is negative which stabilizes, in a thermodynamic sense, the formation of hydrogen bonds at low temperatures. All these concepts were used in a Monte Carlo simulation employed to solve the model. The output of the simulation is the density of states which was used to compute various thermodynamic properties of the model including the phase diagram.
1.3 Literature Review

1.3.1 Experimental Work

Although computer simulation studies of beta sheet formation dominate the field currently,\textsuperscript{5,12,19,20} experimental work is truly needed. However, due to the structural complexity of proteins, the experimental analysis of beta hairpin folding utilizes peptide fragments as opposed to entire proteins.\textsuperscript{12,19-26} One of the more commonly studied peptide segments is the 16 residue, C-terminal segment of protein G.\textsuperscript{19,20,23-29} This particular segment demonstrates a sharp transition while still being small enough for detailed experimental analysis as well as computer simulations. It also has a tryptophan residue buried in the core of its native structure allowing fluorescence to be used to analyze the folding process.

Although some authors have designed peptide segments for their own purposes, most of them have the same basic properties as the aforementioned segment; about 16 residues and a hydrophobic core.\textsuperscript{19-29} When designing a protein, there are several factors that need consideration before one can obtain a beta sheet. First, the hydrophobic core must be placed in the correct location so that the loop will be the desired size. Then residues with charges are added to prevent aggregation of the molecules which is sometimes a problem with protein synthesis. Next, the remaining residues are picked with two things in mind; some residues intrinsically form beta hairpins and some enhance the $^1$H NMR resonances of the molecule. Finally, the actual turn residues are synthesized to promote beta hairpin formation for either the parallel or antiparallel conformation.\textsuperscript{19-26}

After choosing the sequence that gives the best chances to fold into a beta sheet, the molecule then needs to be synthesized. To synthesize these peptide segments, the
authors separate them into parts. First, the mid-section of the segment is made through a process that involves solution and solid phase synthesis methods. Next, standard Fmoc-based procedures have allowed authors to create the “lower” strand. Next, more Fmoc-based procedures are used to create the “upper” strand. These peptide fragments were then cleaved from the surface they were created on, and the protective side chain groups were removed, usually in the same step. The samples were then purified, and MALDI mass spectrometry was used to confirm the structure. An example protein chain developed by Stanger and Gellman is as follows: Ac—Ser—Lys—Phe—Ile—Gln—Val—D-Pro—DADME—Arg—Thr—Tyr—Leu—Val—Lys—Ac. Where the typical three letter abbreviation for residues is used, see Appendix I, and DADME is their designed residue to promote hairpin turn formation.

After the synthesis several techniques are used to determine if the protein fragments had actually formed beta sheets. Because nuclear Overhauser effect, NOE’s, between residues is commonly accepted as the determining factor for anti-parallel beta sheets, they utilized the same method for this analysis. For example, Stanger and Gellman found 22 total NOE’s between nonadjacent residues for their specific protein, all of which fit the pattern corresponding to parallel hairpin formation. Figure 1.3.1 shows five of the backbone CαH—NH NOE’s (the solid arrows). These indicate parallel beta sheet formation. Alpha proton chemical shift has also been used in order to confirm the parallel beta sheet structure. When a residue is part of a beta sheet strand its ΔH shifts downfield relative to its position as a random coil. In comparison, participating in an alpha helix causes the shift to move upfield. Since the values of ΔH are greater than zero for nearly all residues in this particular example, and are quite high for the four inner
residues, the data shows that the chain has a high affinity for beta sheet formation, especially around the turn. To test these methods of analysis authors sometimes create residues that do not allow turns to form and utilized the same analysis techniques. They then compare the results to prove they have formed beta hairpins.\textsuperscript{19}

These experiments help to demonstrate the necessity for simulations. As can be seen, creating proteins in a synthetic setting is quite tedious, and does not offer much information in the way that the structures fold.

1.3.2 Molecular Dynamics

Recently, Pande and collaborators have published a paper pertaining to beta sheet folding.\textsuperscript{29} They, as well as many other molecular dynamic researchers,\textsuperscript{15,16,19-29} decided that since the length of time required to simulate a protein, even one of short length, using molecular dynamics was extremely high, it would be better to take an already formed beta hairpin and simulate it unfolding due to an increase in temperature. The protein segment they chose was one that is well studied in the field due to its highly structured native state,\textsuperscript{19,20,24-29} protein G or 1GB1 in the protein database. They specifically used the C-terminal domain with the sequence: GEWTYDDATKFTVTE (using the single letter notation for each residue as shown in Appendix I). The authors placed this in a 60 angstrom diameter sphere surrounded by water. They then pre-equilibrated the system at 300 K. This is done to both speed up the simulation as well as to make sure that the conformation of the system is actually both viable and realistic. Each time set was two femtoseconds. To demonstrate that the system was stable to begin with, the authors
simulated the protein for 1 nanosecond at 300 degrees and 3 nanoseconds at 350 degrees, the hairpin remained in its native state with only a small amount of fraying at the ends.

To classify the conformations, Vijay and Rokhsar used two criteria.\textsuperscript{29} They use the hydrophobic core size, $R_{\text{core}}$, and the number of hydrogen bonds, $N_{\text{HB}}$. Hydrogen bonds were considered to exist if and only if the carboxyl oxygen from one amino acid and the amide hydrogen from another nonadjacent amino acid are within 2.5 angstroms of one another. $N_{\text{HB}}$ is the number of backbone hydrogen bonds excluding the ends of the strands. $R_{\text{core}}$ is the radius of gyration of the three heavy atoms of the aromatic side chain that make up the hydrophobic cluster inside the hairpin. This term represents the compactness or solvation of the cluster. If the alpha carbons are within a certain distance, 6.5 angstroms, then a contact is said to exist.

Many unfolding pathways were determined ranging in temperature from 350 degrees to 900 degrees Kelvin, and up to 3 nanoseconds in length. These pathways were grouped into transitions between four states; F, the frayed folded state, H, no secondary structure but maintaining the hydrophobic core, S, the partially solvated hydrophobic core, and U, the unfolded state with no core or hydrogen bonding. The probability each chain will be in each conformation is shown in figure 1.3.2. Each step could possibly move back and forth between the neighboring steps especially at higher temperatures. Occasionally one step might be skipped in the transition, but skipping directly from F to U was never observed.

The authors concluded that this particular beta hairpin folds by first forming the core and then forming the hydrogen bonds starting near the turn and then moving down the
strands in a so-called zipper-like fashion. This conclusion is also supported by other experiments.\textsuperscript{5, 19-29}

1.3.3 Monte Carlo

One of the most interesting pieces of work on simulations of beta sheet folding is a paper by Dimitrievski et. al.\textsuperscript{18} This paper discusses a 3-D lattice Monte Carlo simulation of beta sheet folding with an emphasis on chain length, folding temperature, and glass-transition temperature. Both hetero and homopolymers were investigated.

This model is a 3-D lattice model with one monomer, in this case amino acid, per lattice site. The protein is a linear sequence of N amino acids placed along the lattice. All amino acids are approximated to be similar residues. The equilibrium structure is designed to depend on monomer – monomer interactions. Solvent interaction is not taken into account. The heteropolymer model takes into account isotropic interactions, whereas anisotropic interactions are used to mimic hydrogen bonding. A unit vector is used to characterize the monomer orientation so that hydrogen bonding can be properly modeled due to its dependence on monomer orientation. Also taken into account in this protein model is the overall chain rigidity. Finally, in the heteropolymer molecule, there are two monomer types, hydrophobic and hydrophilic. This allows for stabilization of the beta sheet folding via hydrophobic interactions. It also allows for three isotropic interactions, hydrophilic to hydrophilic, hydrophobic to hydrophobic, and hydrophilic to hydrophobic.

The next part of the model deals with the most important interaction in beta sheet formation, hydrogen bonding. The model states that a hydrogen bond can only occur
when two residues are nearest neighbors (NN) in space but not NN in the polypeptide chain. Also, hydrogen bonds require a certain orientation of the two residues relative to one another. If the residue orientation is tied to its physical direction, the protein will form the most compact state possible, bypassing secondary structures altogether. Although some proteins exhibit this, alpha helices and beta sheets also form naturally. This led the authors to allow the hydrogen bonding orientation to be independent of the monomer’s physical position.18

The key to this model is the overall protein energy equation.

\[
E_{\text{tot}} = \sum_{|i-j|\leq3} e_{ij}^{\alpha} \delta(r_{ij} - a) + \sum_{|i-j|\leq3} e_{ij}^{\alpha} \delta(r_{ij} - a) + \sum_{i=1}^{N-1} A(s_{i}s_{i+1}) + \varepsilon_k n_k ,
\]

Eq. 1.3.1

Where \( r_{ij} \) is the distance between monomer i and monomer j, and \( a \) is the lattice spacing. The first term represents the isotropic monomer-monomer interactions. The second term is the energy contribution of hydrogen bonds in topological contacts. Hydrogen bonds only occur when one monomer is directed toward the other monomer, and the hydrogen bonding orientation vectors must be parallel. The third term is the energy based on the spin-spin interaction of the monomers; these interactions determine whether or not the beads attract each other. And finally, the fourth term is the penalty for kink formation. Kink formation in this paper occurs when the angle between monomers is greater or less than the ideal angle set by the authors.

In the heterogeneous model, there are two types of monomers. Monomer A is hydrophobic. Monomer B is hydrophilic. This is intended to reproduce hydrophobic residues being contained inside the hydrophilic residues, which is the initial method of stabilization in the formation of beta sheets.
The moves used in this model are relatively straightforward. A monomer is chosen at random. If it is an end bead, a neighboring lattice site is chosen for an end move. If the monomer is a bead in the middle of the chain, the residue has the possibility of performing either a corner move or a crankshaft move. In a corner move, the bead is moved from one corner to another corner. In a crankshaft move, the residue is rotated around the chain in a direction that is chosen randomly. If the move would cause two monomers to overlap it is rejected. If no overlap occurs, then the energies of the previous and current state are calculated. The move is accepted with the probability given by the Metropolis rule mentioned above. After the move is accepted or rejected, the orientation of the hydrogen bond of one of the monomer is chosen at random for a movement trial, without changing the position of the monomer.

There are several conditions used to implement this model. First, the solvent is not treated explicitly, and the only consideration is the hydrophobic and hydrophobic interactions. Because of this, the energetic parameters should be smaller than they would typically be. The overall free energy difference between the denatured states and the native states is usually low, often between 5 and 15 kcal/mol. This led Dimitrievski et. al.\textsuperscript{18} to assume that the model energetic parameters should be approximately equal to or less than $k_B T$.

For the homogeneous protein chain condition, the model changes only slightly. First, there are no isotropic monomer-monomer interactions because there is only one kind of monomer and no simulation of a solvent. With these conditions, the conformation depends on kink formation and hydrogen bonds alone.
As previously stated, the beta sheets were investigated at multiple lengths. Below is a list of turns versus chain length for the homopolymer proteins.

a. $8 < N < 20$  -One turn.

b. $21 < N < 26$  -One or two turns equivalent.

c. $27 < N < 40$  -Two turns.

d. $40 < N < 50$  -Two or three turns equivalent.

e. $N > 50$  -Three turns.

As shown, the more amino acids in the chain, the more turns there are in the chain.

Heterogeneous proteins show different results. With monomer A being attracted to any other monomer A, strategically introducing monomer A will stabilize these conformations. Figure 1.3.3\textsuperscript{18} shows two different possible conformations, one with the A monomers (filled circles) on the periphery, and one with the A monomers in the center. Sequence 1, (a) from Figure 1.3.3, and Sequence 2, (b), are native for $N=15, 18, 21, 24, 27, 30, \text{ and } 33$ as verified by running $3*10^9$ Monte Carlo Steps (MCS). For longer chains, native states with more than two turns only become favorable when $N$ is greater than 100.

One of the properties investigated was the temperature dependence on the folding time of the protein. Folding time is defined by the authors as the mean first passage time from the initial state to the native state. As expected, the folding time is relatively fast at intermediate temperatures as can be seen in Figure 1.3.4.\textsuperscript{18} At high temperature, the folding time is longer, due to the fact that a higher temperature causes the hydrogen bonds to be relatively weak and unstable. It is also longer at lower temperatures due to the chain being trapped in one of many metastable states. As expected, the ideal temperature interval for folding changes with a change in $N$. As $N$ increases, the
temperature interval decreases rapidly. Larger N increases the number of disordered states, thus causing this phenomenon. Interestingly enough, the fastest folding temperature has a relatively weak dependence on N.

Two other properties investigated in this paper are $T_{\text{Fold}}$ and $T_G$. $T_{\text{Fold}}$ is defined as the temperature at which the protein occupies the native state at least 50% of the time. $T_G$ is defined as the temperature at which the folding time is the average of the maximum and minimum folding times. If $T_{\text{Fold}} / T_G > 1$, the protein is considered to be a good candidate for beta sheet folding.

The dependence of folding time on protein length $N$ has been established as a power law:

$$T_f \propto N^\lambda$$  \hspace{1cm} \text{Eq. 1.3.2}

For both sequences $\lambda$ was found to be in the range of 5.2 \pm 0.8, this value is relatively large and therefore is not considered to be realistic by the authors. For smaller values of $N$ there was a worse fit due to the sensitivity of the chain to the heterogeneity. The equation needs to be adapted to become a realistic representation because of the large value of lambda. The following equation was then proposed:

$$T_f = T_f^m \left( \frac{N}{N_m} \right)^\lambda$$  \hspace{1cm} \text{Eq. 1.3.3}

where $N_m$ is the minimum value of $N$ before the power law breaks down. In this case, the value is 6 because below this number, beta sheets cannot form.

In conclusion, for this system, beta sheets form if directed hydrogen bonds dominate the isotropic interactions. It was determined that, at the most reasonable temperature, the folding time depends on the chain length according to a power law
relationship; with lambda around 5.2. The way to improve this model would be to model the protein in real space and to use the Wang-Landau algorithm as opposed to the Metropolis method.

Another pertinent paper in this field is on the topic of alpha helixes. This paper written by Varshney et. al. deals with a 3-D, real space model using the Wang-Landau algorithm to predict the alpha helix to random coil transition. The model used in this paper was a coarse-grained model which coupled the semiflexibility of the protein backbone with the helix-coil transition to determine the conformational and thermodynamic properties of the protein. These properties are determined using statistical mechanical formulas.

The simulation was started by randomly generating a freely rotating chain of beads, each of which represents a residue, or amino acid. The bond length is set at 1.53 in arbitrary units, and the bond angle was set at 109.5 degrees. The first bead was set at the origin, the second bead was set on the x-axis, and the third was set at an angle of 109.5 degrees from the second in the x-y plane. This chain is then randomly moved using a pivot function which takes a random bead and rotates the chain following that bead randomly to one of any 64 dihedral angles. These dihedral angles are used to determine the state of every bead. A geometrical property called torsion depends on two consecutive dihedral angles and if this property is within a certain tolerance of the torsion of the helix, then the bead is said to be a helical bead. Hydrogen bonds are also a determining factor for helical conformations. If there is no hydrogen bond to support the helical structure, then the bead is still considered a random coil bead. The conformation of the chain is then accepted or rejected based on the previously mentioned Wang-Landau method. The
histogram and density of states arrays are populated and then the process is repeated. The two variables used for the arrays are the number of helical beads and the number of interfaces between helical and coil beads.

Results from this work were derived from the density of states array obtained. These calculations were based on the equations given by Wang and Landau.\textsuperscript{31} The partition function, Helmholtz free energy, internal energy, entropy and heat capacity were all evaluated. These results were also studied with respect to changes in chain length. The work in this thesis follows along these lines closely.

1.4 Monte Carlo Computer Simulations

Many simulation methods are employed in the analysis of protein folding.\textsuperscript{31-39} All of these simulation methods require a large amount of time to converge on a solution and only provide glimpses of the system. Molecular dynamics methods are used to some extent but are very computationally intensive; because they include every atom of the protein; thus, the size of the chain that can be studied in a reasonable period of time is severely limited.\textsuperscript{39-43} These reasons lead many researchers in the field to use Monte Carlo simulations. The main Monte Carlo simulation used for the last fifty years has been based on the Metropolis method.\textsuperscript{31,44} There have been many modifications to it to accomplish specific purposes.\textsuperscript{44,18} One of the more recent alternatives to Metropolis is the so-called Wang-Landau method.\textsuperscript{31} This method has one major advantage over Metropolis in that it does not get trapped in local energy minima and can predict the configuration of the model with the lowest overall free energy.\textsuperscript{31}
1.4.1 The Metropolis Algorithm

The Metropolis algorithm has been used extensively since it was first discovered. The algorithm was designed to perform equation of state calculations. Metropolis et al. created a system with N particles in any configuration within a defined box. Then, they moved each particle around a square by some random amount. Unlike previous models, they then weighed each new conformation with a certain probability and accepted or rejected the move. Previous algorithms only gave each new conformation a specific statistical weight after the new conformation was accepted.

Several assumptions were made in order to simplify the model. First, typical liquid theory assumptions were made. These include assuming classical statistics as well as assuming the potential field of a molecule to be spherically symmetric. Metropolis et. al. also considered only two-body forces. Finally, they assumed the temperature and the density of the system to be unrestricted.

The model starts with N particles in any configuration, specifically on a regular lattice. The particles are then moved one by one using the formula:

\[
\begin{align*}
\theta_{\text{new}} &= x + \alpha \epsilon_1 \\
y_{\text{new}} &= y + \alpha \epsilon_2
\end{align*}
\]

Eq. 1.4.1

Where \( \alpha \) is the maximum allowed displacement, typically it is one half of the total length of one side of the square system. The variables \( \epsilon_1 \) and \( \epsilon_2 \) are random numbers that range from -1 to 1. The energy of the whole system, \( E \), is calculated for each new configuration and compared with the previous conformation. Any negative change in the energy causes the move to be accepted automatically in order for the system to find the lowest energy state possible. If the change in energy of the system is positive then the move is accepted with a certain probability, specifically:
Where $\varepsilon_i$ has a range from 0 to 1. If a move is rejected, then the system returns to the previous configuration and another move is then proposed. Because this system allows for any state to be reached from any other state, it is ergodic and, on average, accepts more moves than it rejects; this implies that the system will approach the canonical distribution.

When the positions of all $N$ particles are known, then the potential energy of the system can be calculated using the equation

$$E = \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} V(d_{ij})$$

Eq. 1.4.3

To calculate the various properties of this system Metropolis et. al. used a canonical ensemble. A general equilibrium value of any property, $F$, can be calculated using the formula

$$\overline{F} = \frac{\int F \cdot e^{\frac{-E}{kT}} \cdot d^{2N}p d^{2N}q}{\int e^{\frac{-E}{kT}} \cdot d^{2N}p d^{2N}q}$$

Eq. 1.4.4

Property $F$ can now be calculated as an average of the overall system and its various states by using the following equation.

$$\overline{F} = \left( \frac{1}{M} \right) \sum_{j=1}^{M} F_j$$

Eq. 1.4.5

In order for this model to work, some conditions should be considered. The most important condition is that $\alpha$ has to be chosen well. If it is made to be too large, all moves
will be rejected, and if made just slightly larger than the particle size, it will cause the model to take a long time to complete. If \( \alpha \) is chosen to be too small, then the configuration of the system will undergo very little change and not give a representative sample of all possible conformations. Also, the hard sphere model should be taken into consideration. By utilizing this condition, some moves will be rejected simply because a sphere is moved into a position where it overlaps another sphere.\(^{31}\)

The modeling method proposed by Metropolis et. al. has a few advantages and disadvantages.\(^{31}\) The main advantage of this method is its simplicity. It is easy to implement and the data can be utilized with only a few simple equations. Also, the Metropolis method is fast relative to molecular dynamics.\(^{40-43}\) With a coarse-grained lattice model, there are considerably fewer calculations required to complete the simulation. The disadvantages of this model can be quite restrictive. If the energy landscape is not relatively smooth or has any well defined local minima then the Metropolis method has difficulty because it accepts any move that decreases the overall energy which causes it to become trapped in the local minimum, and therefore cannot explore the entire phase space. If figure 1.4.1 is an energy landscape, then the Metropolis method would have problems moving from the local minimum on the left side to the overall minimum on the right hand side because the probability of accepting moves with an increase in energy is relatively low compared to accepting the moves that maintain the local minimum. Another disadvantage of this method is that after initializing the positions of all \( N \) particles, the simulation must be run for several cycles to remove any regularity inherent in the system caused by placing the particles on the lattice.\(^{44}\)
1.4.2 The Wang Landau Algorithm

The Wang Landau algorithm is a fairly recently addition to the field. It basically estimates a density of states of the model using the random walk approach which produces a flat histogram in the energy space of the overall system. Because this method is derived directly from statistical mechanics principles, specifically it is based on multi-canonical ensembles, there are no real assumptions.

The method starts by generating a random configuration in space. This configuration has an energy that belongs to an energy window called a bin. From this configuration, the system is moved randomly. Each move is accepted or rejected based on a probability given by the reciprocal of the value of the density of states, $1/\{g(E)\}$, in that particular energy bin. After each state is visited, the bin corresponding to that conformation is incremented in the histogram by one, and incremented in the density of states array by a modification factor $f$, which is always greater than 1. After several iterations the histogram is checked for overall flatness. An overall tolerance is chosen so that if each bin value is within 20% of the average value, then the histogram can be considered flat. This value is determined based on the length of time available for the experiment, higher tolerance, and the accuracy wanted, lower tolerance. After the histogram is found to be flat, it is reset to zero and the modification factor is decreased using any function which monotonically decreases to one. In this case, the square root of the previous modification factor is used. The simulation then repeats the process with the new $f$, and can therefore hone the density of states to produce more accurate results with each following iteration.

The probability used to either accept or reject any move is as follows:
\[ p(E_1 \rightarrow E_2) = \min \left[ \frac{g(E_1)}{g(E_2)}, 1 \right], \quad \text{Eq. 1.4.6} \]

where \( E_1 \) is the overall energy before the move and \( E_2 \) is the energy after the move. If the move is accepted then \( E_2 \) becomes \( E_1 \) and the process continues. After obtaining the density of states for a system, several thermodynamic properties may be obtained. The partition function \( Z \)

\[ Z(E) = \sum_E g(E) e^{-\beta E}, \quad \text{Eq. 1.4.7} \]

the free energy

\[ F(T) = -k_B T \ln(Z) = -k_B T \ln \left( \sum_E g(E) e^{-\beta E} \right), \quad \text{Eq. 1.4.8} \]

internal energy

\[ U(T) = \langle E \rangle_T = \frac{\sum E \ast g(E) e^{-\beta E}}{\sum g(E) e^{-\beta E}}, \quad \text{Eq. 1.4.9} \]

entropy

\[ S(T) = \frac{U(T) - F(T)}{T} \quad \text{Eq. 1.4.10} \]

And finally, the heat capacity

\[ C(T) = \frac{\langle E^2 \rangle_T - \langle E \rangle_T^2}{T^2} \quad \text{Eq. 1.4.11} \]

There is one main consideration when utilizing the Wang-Landau algorithm.\(^{31}\) The range of the arrays has to be carefully investigated so that all possible values are within the overall range. Also, the bin size greatly affects the success of the simulation. If the bins are set too small, there will be too many bins to properly populate, and the
simulation will require a substantial amount of time to run until completion. When the bins are set too large, the data collected will not have the accuracy required to truly utilize the simulation and the error will be a hindrance on the results.

As with all simulation methods, Wang-Landau also has its advantages and disadvantages.\textsuperscript{31} Because the density of states is obtained directly, the Gibbs free energy and the entropy of the system can be directly calculated. Another advantage of this model is that it is efficient and accurate over wide temperature ranges, and it allows for larger sized systems than the ones that are used in the method proposed by Metropolis et. al.\textsuperscript{31,44} Finally, the biggest advantage of Wang-Landau over Metropolis is the fact that since the simulation visits conformations based on the relative density of states of each bin as opposed to being based on the actual energy of the system, the simulation is not trapped in a local energy minima.\textsuperscript{31} Because the system is independent of temperature, only one run is required to obtain data for all temperatures. The main disadvantage of this method is the overall difficulty of implementing the method when compared to Metropolis.
Figure 1.1 Parallel Beta Strands
Figure 1.2 Anti-Parallel Beta Strands
Figure 1.3.1 The NOE’s for the backbone CαH—NH are represented by the solid arrows. These indicate a parallel orientation of the beta strands.
Figure 1.3.29 Probability that the peptide chain will be in any given conformation based on the number of hydrogen bonds and the radius of gyration.
Figure 1.3.3 The Monomer A (Filled) is used to stabilize the strands in specific configurations by preferably bonding with other A monomers and dictating the location of the turns.
Figure 1.3.418 Histogram Distribution of First Folding Time
Figure 1.4.1 Energy landscape example
2.1 Physical Foundations

The physical foundations of the model are based on previous works.\textsuperscript{18,30} The model is a coarse-grained, freely-jointed chain model. Each residue of the chain is modeled as a hard sphere of a diameter similar to the length of an average amino acid. All residues are identical in order to produce a homopolypeptide. The bond angles between spheres are set at 120 degrees. If the angle between the beads deviates from this value then there is an energetic penalty. If two beads form a hydrogen bond then the energy of the current conformation is decreased.

2.2 Chain Generation Algorithm

The program, Appendix II, starts by generating the conformation of the homopolypeptide. For simplicity, the program uses the bead size as a unit length. This means that the bead diameter is equal to one and all other distances and lengths are relative to this length. Each bead also has 2 angles associated with it. The bond angle, $\theta$, is the angle between 3 consecutive beads. Bond angles are calculated from the spatial positions of the beads. The angle is assigned to each bead except for the first and last beads of the chain which cannot have one due to the lack of a third bead to determine the angle. The other angle, the dihedral angle $\phi$, is more complicated. It utilizes 4 consecutive
beads, the first two and last two are compared as two vectors when looking down the line between the middle two. It is calculated from the spatial positions of the beads as well using the following equation

\[ \phi = \cos^{-1}\left(\frac{a \cdot c}{\sqrt{(a \cdot a)(c \cdot c)}}\right) \]  

\text{Eq. 2.1}

Where \(a\) is the vector product of the vector formed by the 2 beads before the selected bead, and the vector formed by the selected bead and the bead after the selected one. The vector \(c\) is the vector product between the vector formed by the selected bead and the previous bead and the vector formed by the selected bead and the next bead. The first bead of the chain is automatically placed at the origin of the coordinate system (Cartesian). This bead is never moved, all other beads are moved. The second bead is also stationary and is placed on the x-axis at exactly one unit away. After these two beads, all other beads are allowed to move freely and have randomly generated locations based on the previous locations of the beads. All beads are 1 unit distance apart; if not then that conformation is rejected. The chain is generated to be \(N\) beads long.

2.3 Energy Function

The energy function of the program serves three functions. First, this function eliminates any chain conformations that are not physically possible. If two neighboring beads are more than 1.0001 units apart, then it rejects the conformation since it indicates that the bond is broken. Also, if two neighboring or not beads are closer together than 0.9999 units then the conformation is also rejected because the beads are overlapping. This latter condition is called the hard sphere model. The second purpose of the energy
function is to calculate the energy required to hold the chain in its current configuration based on bond angles. This term is called the kink energy and is based on the assumption that each bond angle should ideally be 120 degrees. If the angles are greater or less than this value then the energy required to deform the chain that amount is calculated using the following formula.

\[
Kink\_Energy = 1 - \cos(\theta - 60) \quad \text{Eq. 2.2}
\]

This energetic penalty fixes the equilibrium bond angle, \(\theta\), at 120 degrees. This energy is then summed up over the whole chain and becomes one of the parameters for the density of states. The third objective of the energy function is to determine which beads are hydrogen bonded to each other. This is done through a series of conditional statements. First, if the beads are already hydrogen bonded then they cannot hydrogen bond to another bead. Second, the beads must be within a certain distance, in this case they must be within 1.66667 units of one another and more than 0.9999 units apart. These distances are based on the estimated size of the beads which are similar in size to an average amino acid. Next, the position of the beads is taken into account. If the beads are above and below one another, or if they are oriented perpendicular to one another then they will not be allowed to form a hydrogen bond. Finally, a vector is determined from the nearest neighbors of the two beads that are potentially hydrogen bonding. This vector must be parallel within a tolerance of 30 degrees to the vector of the other bead in all three directions for the condition of hydrogen bonding to be satisfied.
2.4 Kink Algorithm

The next part of the program deals with moving the beads of the homopolypeptide chain into different conformations. The first such move is called the kink move. This move rotates one bead around an axis created between its two neighbors without moving any other bead on the chain. Slight conformational alterations are performed with this method. First a random bead is chosen from between the two end beads. The end beads cannot perform this move because of a lack of the necessary axis to pivot around. Next a random angle, $\delta$, is chosen. The bead is then pivoted around the axis by the random angle using the following equation.

$$r_{\text{new}} = n(n \cdot r) + [r - n(n \cdot r)]\cos(\delta) + (r \times n)\sin(\delta)$$  

Eq. 2.3

Where $r$ is the vector between the bead being moved and the previous bead, and $n$ is the vector between the previous bead and the bead after the bead being moved. After the move is performed the function recalculates the bond angle and dihedral angles that were affected.

2.5 Pivot Algorithm

Another type of move in this program is the pivot move. The purpose of this move is to obtain a large conformational change. As in the previous move, a random bead is selected to perform the move. The first two beads and the last bead are excluded due to their lack of dihedral angles. Also, as before, an angle is chosen at random. The chosen bead and the beads before it on the chain remain stationary while the rest of the chain rotates around the dihedral axis. After the move, the function must recalculate the
affected beads new dihedral angles including the beads that were 3 beads before the move. This is done because the dihedral angle of each bead depends on 4 beads total.

2.6 Histogram

The Wang Landau algorithm requires use of two arrays, the histogram array and the density of states array. These arrays have indices of the two factors dictating the energy of the current configuration. In this case, these factors are the number of hydrogen bonds and the kink energy. The number of hydrogen bonds is always an integer, and so is divided into bins for each integer from zero to the maximum, N/2. The kink energy is a non-integer value and is divided up into bins of equal size. In this case each bin has a range of 1, as in all values from 10 to 11 are placed in one bin. The maximum value is set based on several trial runs. It turns out that this value is basically twice the number of beads, N. The histogram, once established, must be populated and tested for flatness. Each time a specific state is visited the corresponding bin is incremented. After many states are visited the histogram is then tested for flatness using a tolerance of twenty percent of the average bin value. The value of each bin that is greater than zero is compared to the average bin value. The zero values are neglected in calculating the flatness because if that particular bin is not populated, it is most likely not physically possible.

2.6 Density of States

The density of states array is similar to the histogram in its indices and in the fact that each time a state is visited it is populated. In contrast, the bins are populated by a
rescaling factor, f. This factor is decreased each time the histogram is determined to be flat. The program runs until it reaches a value approximately equal to 1. The density of states array also dictates the probability a state is visited. This array is the purpose of the program, after obtaining these values the thermodynamic properties can then be computed.

2.7 Program Utilization

All of these elements are put together in the main function in the program. This program utilizes a random number generator. The one chosen is reported to give the most numbers randomly before repeating the sequence, and is called the Mersenne Twister random number generator. Every time a random bead, angle, or position is generated; the random factor comes from this function.

As previously stated, first the homopolypeptide chain must be generated. The chain is generated and then tested for feasibility using the energy function. If the chain is rejected for any reason, the whole chain is regenerated and retested. After generation, the chain then begins moving. First, the kink move is performed on the chain. This move is used nine times for every pivot move. The purpose of this is because the kink move affects fewer beads, it is a more local move, and also is more likely to occur in real protein folding. The pivot move is a larger move and requires more calculations so it is used less often. After each move the energy function is run. This is done to eliminate any unfeasible conformations and to determine the bin that the conformation belongs to if it is acceptable.
The next step is the actual implementation of the Wang Landau algorithm.\textsuperscript{31} This is also done after each individual move and determines whether the move is accepted or rejected. The Wang-Landau function takes two numbers into account, the value of the bins in the density of states array corresponding to both the previous conformation and the current conformation. By comparing these numbers with a random number generated, the bin with the lower value is chosen with more frequency based on the overall logarithmic difference between the values. If the move is rejected, all of the current conformations values, everything from the location of the beads to the bond angles, are changed back to the previous values. Then, the density of states array and the histogram array are both incremented for the previous conformation. If the move is accepted, then the previous conformation is replaced by the current conformation, and the bins in both arrays are incremented for the current conformation.

After every 1,000,000 conformations the histogram is checked for flatness as described above. If it is not flat then the program performs another one million conformations before rechecking. When the histogram is finally considered flat, the density of states at that point is printed to a file. The incrementing factor for the density of states array is then decreased. In this program \( f \) is decreased by setting it equal to the square root of itself. The initial value of \( f \) is 2.71828, i.e. the value of \( e \). After this, the program is rerun starting after the generation, as there is already a chain, and run over and over. The purpose of this is to refine the density of states by smaller and smaller amounts. The program currently is set to run until \( f = 1.0000001 \). This takes approximately 25 iterations.
The data obtained for this work started with a simplified version of the program described above. The simplification of this simulation was that instead of calculating the density of states only the histogram array was populated. The purpose of this preliminary simulation was to determine the feasible bins. These bins are the ones used in the final program, eliminating the possibilities that are impossible or unrealistic. The final simulation only checks the histogram for the determined feasible bins, and also only calculates the feasible bins in the density of states array. This eliminates the problem of a state being visited for the first time at a late point in the program; which would distort the Wang Landau\textsuperscript{31} algorithm and cause the program to become trapped in that state. Any states visited that are not included within the feasible bins were recorded in alternate files so that they could be analyzed later.
CHAPTER III

ANALYSIS

3.1 Preliminary Data

The first set of data obtained was that of the feasible bins. The program was run until all bins with a non-zero value were visited more than five million times. Simulations were also run for up to twenty million times, but the resulting data yielded bins that were not regularly populated in the runs of the complete simulation. The extra bins disrupted the Wang Landau algorithm. With this information, the data from the simulation runs of five million were used. As seen in Table 1 below, the highlighted bins are the ones that were determined to be feasible. Each bin represents the range of kink energy values between the labeled column and the column before it, and also how many hydrogen bonded beads exist in that conformation.

Table 3.1: Feasible Bins

<table>
<thead>
<tr>
<th>Kink Energy</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
<th>16</th>
<th>17</th>
<th>18</th>
<th>19</th>
<th>20</th>
</tr>
</thead>
<tbody>
<tr>
<td>H-Bonded Beads</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>9</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
It was determined that there are only 29 feasible bins for a 10-bead chain. Since each bead can only form one hydrogen bond, only even numbers of hydrogen bonded beads occur. Due to the lack of dihedral angles for the first bead and the last two beads, only seven beads are available for hydrogen bond formation. At least two of these seven beads form the turn of the hairpin; thus, leaving only five potentially hydrogen bonded beads which allows for only a maximum of 4 hydrogen bonded beads. The kink energy ranges from five to sixteen for no hydrogen bonded beads but, as the number of hydrogen bonds increases, the window of kink energies decreases. As the peptide segment becomes more structured there is less freedom of movement and, therefore, fewer feasible bins.

After obtaining the feasible bins, the final simulation was run. Every time the histogram was determined to be flat, the density of states array was printed to a file along with the current value of the factor f. In this way the progress could be monitored. Any state visited that was not in a feasible bin was recorded in another file for later interpretation. The densities of states determined by the program for 10 beads and 16 beads are depicted in the figures 3.1A, 3.1B and 3.12. Lambda is the kink energy and H-bonds is the number of hydrogen bonds. Figure 3.1.A shows a top down view of the density of states illustrating the populated bins. Figure 3.1.B shows a side view demonstrating the natural logarithm values of the density of states on the vertical axis. The density of states array does not show a large change between bins, but because these are relative values, it still yields enough data for analysis.
3.2 Data Analysis Program

Using the newly determined density of states as the input, several thermodynamic properties were calculated using the program from Appendix III. First, we computed the energy of the system from the density of states.

\[ E = (C \cdot HB + \alpha \cdot \lambda_K) \quad \text{Eq. 3.1} \]

where HB is the number of hydrogen bonds and C is the energy that corresponds to the formation of the hydrogen bond; this contribution is negative. For calculation purposes C was set equal -650, the reason for this choice is explained below. \( \lambda_K \) is the kink energy and \( \alpha \) is the stiffness factor which is always positive and dictates the relative value of the kink energy in relation to the hydrogen bonds. Both \( \alpha \) and C are variable factors. Their value is determined by modifying the model so that it predicts realistic values for various transition temperatures.

After determining the energy, the partition function can be determined as follows.

\[ Z(\alpha, T) = \sum_E g(E)e^{-\beta E}, \quad \text{Eq. 3.2} \]

where \( g(E) \) is the density of states and \( \beta = 1/(k_B*T) \). The value of \( \alpha \) was varied from 1 to 1000 and the temperature, T, was varied from 25 K to 625 K. The resulting data is displayed in Figure 3.2. The dominant contributions to the partition function occur in the low end of both temperature and \( \alpha \). The partition function is then used to calculate the Helmholtz free energy.

\[ F(\alpha, T) = -k_B T \ln(Z) = -k_B T \ln\left( \sum_E g(E)e^{-\beta E} \right) \quad \text{Eq. 3.3} \]
The results are shown in Figure 3.3. As the temperature increases the free energy decreases. The overall free energy landscape is smooth. The next variable calculated is the internal energy.

\[
U(\alpha, T) = \left\langle E \right\rangle_T = \frac{\sum_E E \cdot g(E) e^{-\beta E}}{\sum_E g(E) e^{-\beta E}}
\]

Eq. 3.4

As seen in Figure 3.4, for values of \( \alpha \) close to 650, there is a clear change in the slope as a function of \( \alpha \), suggesting the presence of a transition that requires further analysis. The next quantity of interest is the entropy which was calculated using the following thermodynamic relationship

\[
S(\alpha, T) = \frac{U(T) - F(T)}{T}
\]

Eq. 3.5

Figure 3.5 shows the entropy dependence on temperature and \( \alpha \). There are a few regions depending on the value of \( \alpha \); each region is characterized by a different conformation of the chain.

The last calculated property is the heat capacity at constant volume, \( C_V \).

\[
C_V(\alpha, T) = \frac{\left\langle E^2 \right\rangle_T - \left\langle E \right\rangle_T^2}{T^2},
\]

Eq. 3.6

shown in Figures 3.6.A and 3.6.B. The heat capacity has a rugged landscape that can be separated into different regions; each of them corresponds to a different, dominant conformation of the chain. The maxima in the heat capacity surface indicate a transition from one state to another one. Figure 3.6.A shows a different view of Figure 3.6.B that clearly shows the maxima separating different conformations of the peptide segment. Utilizing this information; a phase diagram can be constructed.
3.3 Phase Diagram Program

The information given by the final calculation from the data analysis program was analyzed further using the program shown in Appendix IV to compute the phase diagram of the model. This program scans the heat capacity for local maxima along both directions, $\alpha$ and temperature. Figure 3.7 illustrates the different regions. The $\alpha$ parameter has been varied so that the correct stiffness can be determined. In reality $\alpha$ is a constant number, so the transitions are shown only by the temperature maxima. Each region indicates a different basic conformation of the peptide fragment ranging from a random coil to a beta hairpin to a fully extended conformation. The uppermost region is most likely the random coil conformation, Figure 3.8, because at high temperature entropy dominates over hydrogen bonds. This was confirmed by visual inspection of several conformations from the uppermost region which includes no hydrogen bonding and higher kink energies which indicate a coil structure. The region at high values of $\alpha$ and low temperatures is most likely a fully extended conformation since the stiffness of the polymer backbone dominates over the formation of hydrogen bonds and entropy. Beta hairpins are most likely formed at the region with mid-ranged values of $\alpha$ and low temperatures. The diagrams shown are only examples of the ensemble of states present in each region. Some of the states have many different, but similar, conformations that satisfy the same conditions. The transition between the ordered beta sheet conformation and the random coil was shifted to be near 300 K. This can be done by adjusting the value of the C parameter. As shown in Figure 3.11, changing the value of C to -275 greatly affects where the transitions occur, but does not affect the shape of the transitions. In this case the transition region for small values of $\alpha$ is near 100K.
The model was also run for 16 beads. As expected there was little difference between the results from the 10-bead system and those seen in the 16-bead system. The density of states is more varied due to the larger number of conformations, but it has the same basic shape (Figure 3.12). The entropy and heat capacities are shown in Figures 3.13 and 3.14. The overall difference is that the 16 bead system has shifted up the transition temperature and sharpened the peaks in the transitions. A C value of -650 was used again to compare the results with the 10 bead system. Figure 3.15 displays the phase diagram for the 16-bead system.
Figure 3.1 a) Density of States: Top View of the 10-bead system
Figure 3.1 b) Density of States: Side View of the 10-bead system
Figure 3.2 Partition Function of the 10-bead system
Figure 3.3 Free Energy of the 10-bead system
Figure 3.4 Internal Energy for the 10-bead system
Figure 3.5 Entropy of the 10-bead system
Figure 3.6 a) Heat Capacity of the 10-bead system: Angle View
Figure 3.6 a) Heat Capacity of the 10-bead system: Side View
Figure 3.7 Phase Diagram for the 10-beads system with a C parameter of -650.
Figure 3.8 Random Coil of the 10-bead system
Figure 3.9 Extended Chain Conformation of the 10-bead system
Figure 3.10 Hairpin Conformation of the 10-bead system
Figure 3.11 Phase Diagram for a 10-bead system with a C parameter of -275.
Figure 3.12 Density of States of the 16-bead system.
Figure 3.13 Entropy of the 16-bead system
Figure 3.14 Heat Capacity of the 16-bead system
Phase Diagram for 16 bead chain (C=-650)

Figure 3.15 Phase Diagram for the 16-bead system with a C parameter of -650
CHAPTER IV

CONCLUSIONS

With a run of 10 beads several things can be concluded. To obtain the desired transition between the random coil and the beta sheet the value of the C parameter must be near -650. The value of $\alpha$ was varied from 1 to 1000 instead of the 0 to 1 window studied by Dimetrievski et. al.\(^{18}\) This model successfully predicts multiple states in 3 dimensional space including extended chains, beta hairpins, and random coils similar to the ones shown in Figures 3.8 through 3.10. These states are the ones expected and are similar to previous works.\(^5,9,22-29\)

The model determines feasible conformations, but does not identify the folding mechanism that real peptides utilize to reach these conformations. Since there is no time element in the simulation, the conformational changes over time cannot be determined. Also, because this is a homopolymer model, no bead-specific changes are considered which, in principle, could take into account the preference of certain amino acid sequences to form beta strands, alpha helices, or even facilitate turns.

Due to the length of time required to run the simulations to completion, only data for 10 and 16 beads could be obtained. With some modification this model can potentially work on much larger or even more complex systems given enough time for the calculations.
CHAPTER V
FUTURE WORK

The model developed in this thesis is not complete; several more steps must be taken to fully understand its implications. The current results require further analysis. Many modifications can be made to the model to obtain more information, as well as improve its ability to model real life systems.

The next step that needs to be taken on the results from both the 10-bead and 16-bead peptide segments is to determine the main conformations for each region of the phase diagram. A simulation must be run to generate conformations for the different values of $\alpha$ and temperature. This would be done by comparing end to end distances and determining which distance occurs most frequently. Afterwards the transitions could be analyzed more thoroughly.

Because time constraints imposed by each simulation, the peptide fragments were only 10 or 16 beads long. In the future, more beads should be modeled to obtain a clearer understanding of the transitions. Specifically, on the smaller scale, the 16 bead model should be compared with the experimental values to determine the accuracy of the results obtained. After this is done, even longer chains could be modeled to form beta sheets or possibly even beta barrels. If the chains are lengthened for this purpose, then the hydrogen bonds per bead must be allowed to be up to two. This needs to be done so that multiple turns can be linked together. Each residue of a protein has the potential to form
at least two hydrogen bonds, one carboxylic end and one amine end. With this new condition beta barrels could even be achieved given enough processing time.

Yet another modification to the model is the addition of multiple bead types. At least two types of beads are required to accurately model chain conformations. These types must be hydrophobic and hydrophilic. The hydrophobic bead interactions help the structures form. They collapse the chain much more quickly than it would normally, placing it into a metastable state where the hydrogen bonds then form. If a hydrophobic interaction was implemented, then it would most likely show up in the phase diagram with more transitions.

The two different conformations of beta hairpins should also be studied. Anti-parallel and parallel sheets should be modeled separately. This should be done by modifying the geometry required for hydrogen bonding in the energy function. With both systems modeled under the same conditions a more direct comparison can be made to better understand how they differ.
REFERENCES


## APPENDIX A

### TABLE OF \(\alpha\)-AMINO ACIDS FOUND IN PROTEINS

<table>
<thead>
<tr>
<th>Amino Acid</th>
<th>Symbol</th>
<th>Structure*</th>
<th>(pK_1) (COO(\text{H}))</th>
<th>(pK_2) (NH(\text{2}))</th>
<th>pKR Group</th>
</tr>
</thead>
<tbody>
<tr>
<td>Glycine</td>
<td>Gly - G</td>
<td>H–CH–COOH</td>
<td>2.4</td>
<td>9.8</td>
<td></td>
</tr>
<tr>
<td>Alanine</td>
<td>Ala - A</td>
<td>CH(_3)–CH–COOH</td>
<td>2.4</td>
<td>9.9</td>
<td></td>
</tr>
<tr>
<td>Valine</td>
<td>Val - V</td>
<td>H(_3)CH–CH–COOH</td>
<td>2.2</td>
<td>9.7</td>
<td></td>
</tr>
<tr>
<td>Leucine</td>
<td>Leu - L</td>
<td>H(_3)CH–CH(_2)–CH–COOH</td>
<td>2.3</td>
<td>9.7</td>
<td></td>
</tr>
<tr>
<td>Isoleucine</td>
<td>Ile - I</td>
<td>H(_3)CH–CH(_2)–CH–COOH</td>
<td>2.3</td>
<td>9.8</td>
<td></td>
</tr>
<tr>
<td>Serine</td>
<td>Ser - S</td>
<td>HO–CH(_2)–CH–COOH</td>
<td>2.2</td>
<td>9.2</td>
<td>~13</td>
</tr>
<tr>
<td>Threonine</td>
<td>Thr - T</td>
<td>H(_3)C–CH–COOH</td>
<td>2.1</td>
<td>9.1</td>
<td>~13</td>
</tr>
</tbody>
</table>
### Amino Acids with Sulfur-Containing R-Groups

<table>
<thead>
<tr>
<th>Amino Acid</th>
<th>R-Group</th>
<th>Structure</th>
<th>pK&lt;sub&gt;a&lt;/sub&gt;</th>
<th>pI</th>
<th>pK&lt;sub&gt;b&lt;/sub&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cysteine</td>
<td>Cys</td>
<td>H&lt;sub&gt;2&lt;/sub&gt;S-CH&lt;sub&gt;2&lt;/sub&gt;CH-CO&lt;sub&gt;2&lt;/sub&gt;H&lt;sub&gt;2&lt;/sub&gt;</td>
<td>1.9</td>
<td>10.8</td>
<td>8.3</td>
</tr>
<tr>
<td>Methionine</td>
<td>Met</td>
<td>H&lt;sub&gt;3&lt;/sub&gt;C-S-(CH&lt;sub&gt;2&lt;/sub&gt;)&lt;sub&gt;2&lt;/sub&gt;CH-CO&lt;sub&gt;2&lt;/sub&gt;H&lt;sub&gt;2&lt;/sub&gt;</td>
<td>2.1</td>
<td>9.3</td>
<td></td>
</tr>
</tbody>
</table>

### Acidic Amino Acids and their Amides

<table>
<thead>
<tr>
<th>Amino Acid</th>
<th>R-Group</th>
<th>Structure</th>
<th>pK&lt;sub&gt;a&lt;/sub&gt;</th>
<th>pI</th>
<th>pK&lt;sub&gt;b&lt;/sub&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aspartic Acid</td>
<td>Asp</td>
<td>HOOC-CH&lt;sub&gt;2&lt;/sub&gt;CH-CO&lt;sub&gt;2&lt;/sub&gt;H&lt;sub&gt;2&lt;/sub&gt;</td>
<td>2.0</td>
<td>9.9</td>
<td>3.9</td>
</tr>
<tr>
<td>Asparagine</td>
<td>Asn</td>
<td>H&lt;sub&gt;2&lt;/sub&gt;N-C-CH&lt;sub&gt;2&lt;/sub&gt;CH-CO&lt;sub&gt;2&lt;/sub&gt;H&lt;sub&gt;2&lt;/sub&gt;</td>
<td>2.1</td>
<td>8.8</td>
<td></td>
</tr>
<tr>
<td>Glutamic Acid</td>
<td>Glu</td>
<td>HOOC-CH&lt;sub&gt;2&lt;/sub&gt;CH&lt;sub&gt;2&lt;/sub&gt;CH-CO&lt;sub&gt;2&lt;/sub&gt;H&lt;sub&gt;2&lt;/sub&gt;</td>
<td>2.1</td>
<td>9.5</td>
<td>4.1</td>
</tr>
<tr>
<td>Glutamine</td>
<td>Gln</td>
<td>H&lt;sub&gt;2&lt;/sub&gt;N-C-CH&lt;sub&gt;2&lt;/sub&gt;CH&lt;sub&gt;2&lt;/sub&gt;CH-CO&lt;sub&gt;2&lt;/sub&gt;H&lt;sub&gt;2&lt;/sub&gt;</td>
<td>2.2</td>
<td>9.1</td>
<td></td>
</tr>
</tbody>
</table>

### Basic Amino Acids

<table>
<thead>
<tr>
<th>Amino Acid</th>
<th>R-Group</th>
<th>Structure</th>
<th>pK&lt;sub&gt;a&lt;/sub&gt;</th>
<th>pI</th>
<th>pK&lt;sub&gt;b&lt;/sub&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arginine</td>
<td>Arg</td>
<td>H&lt;sub&gt;2&lt;/sub&gt;N-CH&lt;sub&gt;2&lt;/sub&gt;CH&lt;sub&gt;2&lt;/sub&gt;CH&lt;sub&gt;2&lt;/sub&gt;CH-C NH&lt;sub&gt;2&lt;/sub&gt;</td>
<td>1.8</td>
<td>9.0</td>
<td>12.5</td>
</tr>
<tr>
<td>Lysine</td>
<td>Lys</td>
<td>H&lt;sub&gt;2&lt;/sub&gt;N-(CH&lt;sub&gt;2&lt;/sub&gt;)&lt;sub&gt;4&lt;/sub&gt;-CH-CO&lt;sub&gt;2&lt;/sub&gt;H&lt;sub&gt;2&lt;/sub&gt;</td>
<td>2.2</td>
<td>9.2</td>
<td>10.8</td>
</tr>
<tr>
<td>Histidine</td>
<td>His</td>
<td>CH&lt;sub&gt;2&lt;/sub&gt;-CH-CO&lt;sub&gt;2&lt;/sub&gt;H&lt;sub&gt;2&lt;/sub&gt;</td>
<td>1.8</td>
<td>9.2</td>
<td>6.0</td>
</tr>
</tbody>
</table>
## Amino Acids with Aromatic Rings

<table>
<thead>
<tr>
<th>Amino Acid</th>
<th>R-Group</th>
<th>Chemical Structure</th>
<th>pKa1</th>
<th>pKa2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Phenylalanine</td>
<td>Phe - F</td>
<td><img src="image" alt="Phenylalanine" /></td>
<td>2.2</td>
<td>9.2</td>
</tr>
<tr>
<td>Tyrosine</td>
<td>Tyr - Y</td>
<td><img src="image" alt="Tyrosine" /></td>
<td>2.2</td>
<td>9.1</td>
</tr>
<tr>
<td>Tryptophan</td>
<td>Trp - W</td>
<td><img src="image" alt="Tryptophan" /></td>
<td>2.4</td>
<td>9.4</td>
</tr>
</tbody>
</table>

## Imino Acids

<table>
<thead>
<tr>
<th>Amino Acid</th>
<th>R-Group</th>
<th>Chemical Structure</th>
<th>pKa1</th>
<th>pKa2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Proline</td>
<td>Pro - P</td>
<td><img src="image" alt="Proline" /></td>
<td>2.0</td>
<td>10.6</td>
</tr>
</tbody>
</table>

*Backbone of the amino acids is red, R-groups are black.*

---

73
APPENDIX B

MAIN PROGRAM

/* Dated 3/06/06
   Freely Jointed Chain, Course-Grain Model of Beta Sheet Folding*/

/* Standard Libraries */
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>
#include <ctype.h>
#include <time.h>

/* CONSTANTS */
#define NB 10                                /* Number of Beads */
#define BL 1.00                               /* Bond Length */
#define HBL 1.667                              /* H-Bond Length */
#define HBtol 0.523598776                      /* H-bond angle tolerance (PI/6)*/
#define HBeng -1.00                            /* Energy gained by creating a Hydrogen bond */
#define kpen 1.00                               /* Energy penalty due to kink formation */
#define kbin 20                                /* Bin size for kink energy */
#define PI 3.141592654                         /* Definition of PI */
#define e 2.71828                              /* The Constant*/
#define X0 0.                                  /* Laboratory coordinates */
#define Y0 0.                                  /* Laboratory coordinates */
#define Z0 0.                                  /* Laboratory coordinates */
#define dihed 6.28318                           /* Number of free rotations*/
#define sqrtsize ((NB-1)*(NB-1))*1000+1       /* Size of sqrt array */

/* Definitions */
struct vector1
{
    double del_x,del_y,del_z;
    double x,y,z;
    double n1,n2,n3;
    double m1,m2,m3;

double xnew, ynew, znew;
    double nr1, nr2, nr3;
};

struct monomers                     /* individual monomer */
{
    double x, y, z;                      /* with cartesian coordinates */
    double phi;                                /* Bond and Dihedral Angle -indices are according to
    Flory- and Bond Length */
    double the, len;
    int HB;                                    /* H-bond yes or no*/
};

struct angle_value
{
    double cosine;
    double sine;
    double acosine;
};

/* GLOBALS */
FILE *DOS, *pre, *acoord, *bcoord, *ccoord;  /* Pointers to data files */
FILE *extra, *extra_1, *extra_2;

/* Initial and final energies for one Pivot move in terms
of integer */
double dist[NB][NB];                         /* Distance between any two beads */
double binf=BL-0.001;                        /* lower limit of bond fluctuation*/
double bup=BL+0.001,x,y,z;                   /* upper limit of bond fluctuation*/
double T[NB][4][4];                          /* Matrix carrying information about Dihedral*/
double TT[NB][4][4];                         /* Matrix carrying information about multiplication
of dihedrals */
double T_bef[NB][4][4];                      /* Matrix carrying information about previous
dihedral */
double TT_bef[NB][4][4];                     /* Matrix carrying information about previous
multiplication of dihedrals */
int kink, pivot;                             /* Random bead picked for move. */
double EtoE;                                 /* End to end distance */

int closecon =0;
int breakcount =0;
int hbonds =0;
int totHB[NB+1];
double kmax =0;
double kmin = 100;
double Asqrt[sqrtsize];

struct monomers polymer[NB]; /* Defines polymer as an array of structures of type monomers */
struct monomers polymer_bef[NB]; /* Defines polymer_bef as an array of structures of type monomers */
struct angle_value AV[36000000]; /* Stores values of cosine and sine of possible didedral angles */
/* struct angle_value AV2[200001]; /* Stores values of cosine and sine of possible bond angles */
struct vector1 line; /* Stores the line kinks are rotated about in kink_move() */

double hist[NB+1][kbin]; /* Array in which histogram has to be updated each time */
double hist_extra[NB+1][kbin]; /* Array in which extra data not anticipated is stored */
double check[NB+1][kbin]; /* Checks to see if data is in correct bin */
double DOSarray[NB+1][kbin]; /* Density of states array */

/*****************************************************************************/
/*------------------- MT CODE -------------------*/
/*****************************************************************************/
/*Mersenne Twister random number generator. 
more info -> http://www.math.keio.ac.jp/~matumoto/emt.html 
sgenrand function seeds the generator, genrand will yeild a random double [0,1] */

/* Period parameters */
#define N 624
#define M 397
#define MATRIX_A 0x9908b0df /* constant vector a */
#define UPPER_MASK 0x80000000 /* most significant w-r bits */
#define LOWER_MASK 0x7fffffff /* least significant r bits */

/* Tempering parameters */
#define TEMPERING_MASK_B 0x9d2c5680
#define TEMPERING_MASK_C 0xefc60000
#define TEMPERING_SHIFT_U(y) (y >> 11)
#define TEMPERING_SHIFT_S(y) (y << 7)
#define TEMPERING_SHIFT_T(y) (y << 15)
#define TEMPERING_SHIFT_L(y) (y >> 18)
static unsigned long mt[N]; /* the array for the state vector */
static int mti=N+1; /* mti==N+1 means mt[N] is not initialized */

/* initializing the array with a NONZERO seed */
void sgenrand(unsigned long seed)
{
    /* setting initial seeds to mt[N] using         */
    /* the generator Line 25 of Table 1 in          */
    /* [KNUTH 1981, The Art of Computer Programming*/
    /* Vol. 2 (2nd Ed.), pp102] */
    mt[0]= seed & 0xffffffff;
    for (mti=1; mti<N; mti++)
        mt[mti] = (69069 * mt[mti-1]) & 0xffffffff;
}

double genrand()
/* generating reals */
/* unsigned long */ /* for integer generation */
{
    unsigned long y;
    static unsigned long mag01[2]={0x0, MATRIX_A};
    /* mag01[x] = x * MATRIX_A  for x=0,1 */

    if (mti >= N) { /* generate N words at one time */
        int kk;
        if (mti == N+1) /* if sgenrand() has not been called, */
            sgenrand(4357); /* a default initial seed is used */

        for (kk=0;kk<N-M;kk++) {
            y = (mt[kk]&UPPER_MASK)|(mt[kk+1]&LOWER_MASK);
            mt[kk] = mt[kk+M] ^ (y >> 1) ^ mag01[y & 0x1];
        }
        for (;kk<N-1;kk++) {
            y = (mt[kk]&UPPER_MASK)|(mt[kk+1]&LOWER_MASK);
            mt[kk] = mt[kk+(M-N)] ^ (y >> 1) ^ mag01[y & 0x1];
        }
        y = (mt[N-1]&UPPER_MASK)|(mt[0]&LOWER_MASK);
        mt[N-1] = mt[M-1] ^ (y >> 1) ^ mag01[y & 0x1];

        mti = 0;
    }
}
y = mt[mti++];
y ^= TEMPERING_SHIFT_U(y);
y ^= TEMPERING_SHIFT_S(y) & TEMPERING_MASK_B;
y ^= TEMPERING_SHIFT_T(y) & TEMPERING_MASK_C;
y ^= TEMPERING_SHIFT_L(y);

return ( (double)y * 2.3283064370807974e-10 ); /* reals */
/* return y; */ /* for integer generation */
}

/*-----------------------------------------------*/
/*--------------- END OF MT CODE ----------------*/
/*-----------------------------------------------*/

/*********************************************/
/* This is the matrix multiplication routine.*/
/* It multiplies two matrices a*b to give c. */
/* In this particular case, it updates the */
/* matrix TT from its previous value and the */
/* value of the matrix T. It is used by the */
/* Pivot algorithm. */
/**/
///
/*********************************************/

void MatrixP(long int CI)
{
    TT[CI][1][1]= TT[CI-1][1][1]*T[CI][1][1]+ TT[CI-1][1][2]*T[CI][2][1]+ TT[CI-1][1][3]*T[CI][3][1];
    TT[CI][1][2]= TT[CI-1][1][1]*T[CI][1][2]+ TT[CI-1][1][2]*T[CI][2][2]+ TT[CI-1][1][3]*T[CI][3][2];
    TT[CI][1][3]= TT[CI-1][1][2]*T[CI][2][3]+ TT[CI-1][1][3]*T[CI][3][3];

    TT[CI][2][1]= TT[CI-1][2][1]*T[CI][1][1]+ TT[CI-1][2][2]*T[CI][2][1]+ TT[CI-1][2][3]*T[CI][3][1];
    TT[CI][2][2]= TT[CI-1][2][1]*T[CI][1][2]+ TT[CI-1][2][2]*T[CI][2][2]+ TT[CI-1][2][3]*T[CI][3][2];
    TT[CI][2][3]= TT[CI-1][2][2]*T[CI][2][3]+ TT[CI-1][2][3]*T[CI][3][3];
}
\[ TT[CI][3][1] = TT[CI-1][3][1]*T[CI][1][1] + TT[CI-1][3][2]*T[CI][2][1] + TT[CI-1][3][3]*T[CI][3][1]; \]
\[ TT[CI][3][2] = TT[CI-1][3][1]*T[CI][1][2] + TT[CI-1][3][2]*T[CI][2][2] + TT[CI-1][3][3]*T[CI][3][2]; \]
\[ TT[CI][3][3] = TT[CI-1][3][2]*T[CI][2][3] + TT[CI-1][3][3]*T[CI][3][3]; \]

FLICTED_TEMPLATE
class /
/** This function generates the first conformation */
/** of the polymer chain randomly. It follows the */
/** Random Flight Model. It initializes the bond */
/** lengths, bond angles, dihedral angles, coor- */
/** dinates of the beads, T and TT matrices.       */
/** */
/** */
/** */
/** */
/** This function generates the first conformation */
/** of the polymer chain randomly. It follows the */
/** Random Flight Model. It initializes the bond */
/** lengths, bond angles, dihedral angles, coor- */
/** dinates of the beads, T and TT matrices.       */
/** */
/** */
/** */
/** */
/*********************************************************************************/

// This function generates the first conformation */
// of the polymer chain randomly. It follows the */
// Random Flight Model. It initializes the bond */
// lengths, bond angles, dihedral angles, coor- */
// dinates of the beads, T and TT matrices.       */
/********************************************************************************/

/* Initialization of the torsional angles */
polymer[0].phi=0.;
polymer[0].the=0.;
polymer[1].phi=PI;
polymer[NB-1].phi=0.;
polymer[NB-1].the=0.;

/* Initialization of the dihedral angles */
for(i=2;i<NB-1;i++)
{
  polymer[i].phi= 2.0*PI*genrand();
}

/* Initialization of the bond angles */
for(i=1;i<NB-1;i++)
{
  polymer[i].the=(PI/3)*(1+2*genrand());
}
/* Initialization of the bond lengths */
for(i=0;i<NB-1;i++)
{
    polymer[i].len=BL;
}

/* Initialization of the T matrix */
polymer[0].x=X0;
polymer[0].y=Y0;
polymer[0].z=Z0;
polymer[1].x=X0+BL;
polymer[1].y=Y0;
polymer[1].z=Z0;

/*for(i=1;i<NB;i++)
{
    arg1=(int)(polymer[i].the*100000*57.29577952);
    arg2=(int)(polymer[i].phi*100000*57.29577952);
    AV_theta_cosine=AV[arg1].cosine;
    AV_theta_sine=AV[arg1].sine;
    AV_phi_cosine=AV[arg2].cosine;
    AV_phi_sine=AV[arg2].sine;
    T[i][1][1]=-AV_theta_cosine;
    T[i][1][2]=AV_theta_sine;
    T[i][1][3]=0.;
    T[i][2][1]=-AV_theta_sine*AV_phi_cosine;
    T[i][2][2]=-AV_theta_cosine*AV_phi_cosine;
    T[i][2][3]=-AV_phi_sine;
    T[i][3][1]=-AV_theta_sine*AV_phi_sine;
    T[i][3][2]=-AV_theta_cosine*AV_phi_sine;
    T[i][3][3]=AV_phi_cosine;
}*/
for(i=1;i<NB;i++)
{
    AV_theta_cosine=cos(polymer[i].the);
    AV_theta_sine=sin(polymer[i].the);
    AV_phi_cosine=cos(polymer[i].phi);
    AV_phi_sine=sin(polymer[i].phi);
    T[i][1][1]=-AV_theta_cosine;
    T[i][1][2]=AV_theta_sine;
    T[i][1][3]=0.;
    T[i][2][1]=-AV_theta_sine*AV_phi_cosine;
    T[i][2][2]=-AV_theta_cosine*AV_phi_cosine;
\[
T[i][2][3] = -AV_{\phi}\text{sine}; \quad T[i][3][1] = -AV_{\theta}\text{sine}\cdot AV_{\phi}\text{sine}; \quad T[i][3][2] = -AV_{\theta}\text{cosine}\cdot AV_{\phi}\text{sine}; \quad T[i][3][3] = AV_{\phi}\text{cosine};
\]

/* Initialization of the TT matrix */
for (i=1; i<4; i++)
{
    for (j=1; j<4; j++)
    {
        TT[1][i][j] = T[1][i][j];
        if (i == j)
            TT[0][i][j] = 1.0;
        else
            TT[0][i][j] = 0.0;
    }
}
for (i=2; i<NB; i++)
{
    MatrixP(i);
}

/* Calculations of the bead coordinates */
for (i=2; i<NB; i++)
{
    polymer[i].x = polymer[i-1].x + TT[i-1][1][1]*BL;
    polymer[i].y = polymer[i-1].y + TT[i-1][2][1]*BL;
    polymer[i].z = polymer[i-1].z + TT[i-1][3][1]*BL;
}

double Energy()


double auxx[NB][NB], auxy[NB][NB], auxz[NB][NB], auxd;
double sum, co, si;
double x, y, z, x2, y2, z2, x3, y3, z3, mag3;
double xj, yj, zj, xj2, yj2, zj2, xj3, yj3, zj3;
double a, b, c, a2, b2, c2, a3, b3, c3;
double dotp, dotp2, dotp3, argum;
double rEnergy = 0.0;
register int i, j;

/*******************************/
/******Close Contact & Chain Break******/
/*****************************/
for (i = 0; i < NB - 1; i++)
{
    for (j = i + 1; j < NB; j++)
    {
        auxx[i][j] = polymer[i].x - polymer[j].x;
        auxx[j][i] = auxx[i][j];
        auxy[i][j] = polymer[i].y - polymer[j].y;
        auxy[j][i] = auxy[i][j];
        auxz[i][j] = polymer[i].z - polymer[j].z;
        auxz[j][i] = auxz[i][j];
        auxd = auxx[i][j]*auxx[i][j] + auxy[i][j]*auxy[i][j] + auxz[i][j]*auxz[i][j];
        dist[i][j] = auxd;
        dist[j][i] = auxd;
        if (auxd < (binf*binf))
        {
            closecon++;
            return 1000;
        }
    }

    if (dist[i][i + 1] > (bup*bup))
    {
        breakcount++;
        printf("\n\n%d %d %f %f BREAK!!!!!! \n", i, i + 1, dist[i][i + 1], bup);
        return 1000;
    }

} /* End to end distance */
EtoE = auxx[0][NB-1]*auxx[0][NB-1] + auxy[0][NB-1]*auxy[0][NB-1] + auxz[0][NB-1]*auxz[0][NB-1];
/******Kink Penalty******/
/****************************/
for (i=1;i<NB-1;i++)
{
    co=auxx[i][i-1]*auxx[i+1][i]+auxy[i][i-1]*auxy[i+1][i]+auxz[i][i-1]*auxz[i+1][i];
    si=sqrt((1-co*co)*3);
    rEnergy=rEnergy+(1-(co-si)*0.5);/* 1-cos(theta-60) The penalty occurs at angles
other than 120 */
}

if(kmax<rEnergy)
{
    kmax=rEnergy;
}
if(kmin>rEnergy)
{
    kmin=rEnergy;
}

/****************************/
/******Hydrogen Bonding******/
/****************************/
for(i=0;i<NB;i++)
{
    polymer[i].HB=0;
}
hbonds=0;
argum=AV[(int)(HBtol*100000*57.29577952)].cosine;
for(i=2;i<NB-2;i++)
{
    for(j=i+2;j<NB-1;j++)
    {
        if ((polymer[i].HB==0) && (polymer[j].HB==0))
        {
            if(dist[i][j]<=HBL*HBL))
            {
                if((polymer[i].phi>2.61799388) && (polymer[i].phi<3.66519143))
                {
                    if((polymer[j].phi>2.61799388) && (polymer[j].phi<3.66519143))
                    {
                        x=auxx[i][i-1];
                        y=auxy[i][i-1];
                    }
                }
            }
        }
    }
}
\[ z = \text{auxz}[i][i-1]; \]
\[ x2 = \text{auxx}[i+1][i]; \]
\[ y2 = \text{auxy}[i+1][i]; \]
\[ z2 = \text{auxz}[i+1][i]; \]

\[ a = y * z2 - y2 * z; \]
\[ b = x2 * z - x * z2; \]
\[ c = x * y2 - x2 * y; \]

\[ xj = \text{auxx}[j][j-1]; \]
\[ yj = \text{auxy}[j][j-1]; \]
\[ zj = \text{auxz}[j][j-1]; \]

\[ xj2 = \text{auxx}[j+1][j]; \]
\[ yj2 = \text{auxy}[j+1][j]; \]
\[ zj2 = \text{auxz}[j+1][j]; \]

\[ a2 = yj * zj2 - yj2 * zj; \]
\[ b2 = xj2 * zj - xj * zj2; \]
\[ c2 = xj * yj2 - xj2 * yj; \]

\[ \text{dotp} = \text{fabs}((a * a2 + b * b2 + c * c2) / (\sqrt{(a * a + b * b + c * c) * (a2 * a2 + b2 * b2 + c2 * c2)})); \]

\[ \text{if} (\text{dotp} \geq \text{argum}) \]
\[ \{ \]
\[ x3 = x2 - x; \]
\[ y3 = y2 - y; \]
\[ z3 = z2 - z; \]
\[ xj3 = xj2 - xj; \]
\[ yj3 = yj2 - yj; \]
\[ zj3 = zj2 - zj; \]

/*\]
\[ x3 = \text{auxx}[i+1][i]-\text{auxx}[i][i-1]; \]
\[ y3 = \text{auxy}[i+1][i]-\text{auxy}[i][i-1]; \]
\[ z3 = \text{auxz}[i+1][i]-\text{auxz}[i][i-1]; */

/*\]
\[ xj3 = \text{auxx}[j+1][j]-\text{auxx}[j][j-1]; \]
\[ yj3 = \text{auxy}[j+1][j]-\text{auxy}[j][j-1]; \]
\[ zj3 = \text{auxz}[j+1][j]-\text{auxz}[j][j-1]; */

\[ \text{mag3} = \text{sqrt}(x3 * x3 + y3 * y3 + z3 * z3); \]
\[ \text{dotp2} = \frac{x_3 \cdot x_{j3} + y_3 \cdot y_{j3} + z_3 \cdot z_{j3}}{\text{mag3} \cdot \sqrt{x_{j3}^2 + y_{j3}^2 + z_{j3}^2}}; \]

\[
\text{if}(\text{dotp2} \geq \text{argum})
\{
\quad a_3 = \text{auxx}[i][j];
\quad b_3 = \text{auxy}[i][j];
\quad c_3 = \text{auxz}[i][j];
\}
\]

\[ \text{dotp3} = \frac{x_3 \cdot a_3 + y_3 \cdot b_3 + z_3 \cdot c_3}{\text{mag3} \cdot \sqrt{a_3^2 + b_3^2 + c_3^2}}; \]

\[
\text{if}(\text{dotp3} \geq \text{argum})
\{
\quad \text{polymer}[i].HB = j;
\quad \text{polymer}[j].HB = i;
\quad /*\text{rEnergy}=\text{rEnergy}+\text{HBeng}; \text{rEnergy} \text{ should be kink energy only (see DOSarray)} */
\}
\]

\[
\text{hbonds} = 0;
\text{for}(j = 0; j < \text{NB}; j++)
\{
\quad \text{if}(\text{polymer}[j].HB != 0)
\quad \quad \text{hbonds}++;
\}
\text{totHB}[\text{hbonds}]++;
\text{return rEnergy};
\]
/ * Printing out which beads are bonded to which */
void Energy1()
{
    int i;
    printf("H-Bonds ");
    for(i=0;i<NB;i++)
        if (polymer[i].HB!=0)
        {
            printf("%d %d    ",i+1,polymer[i].HB+1);
        }
    printf("\n");
    for(i=0;i<NB;i++)
        if (polymer[i].HB!=0)
        {
            printf("Dihedral[%d]= %f ",i+1,180*polymer[i].phi/PI);
        }
    printf("\n");
}

/****************************/
/****************************/
/****************************/
/**** Kink move function ****/
/****************************/
/****************************/
/****************************/
void kink_move()
{
    double x,y,z,x2,y2,z2,x3,y3,z3;
    double a1,a2,a3,c1,c2,c3,d1,d2,d3,ac,ab,bc;
    double mag1,mag2,mag3,ndotr;                   /*Magnitudes of vectors used for calculations.*/
    int delta;
    int i,j;
    int arg1, arg2;
    double AV_theta_cosine, AV_theta_sine, AV_phi_cosine , AV_phi_sine;

    kink=(int)(1+genrand()*(NB-3)); /*Random bead chosen so that it can accomplish move and have the dihedral recalc.*/
delta=(int)(36000000*genrand());

if (kink>1)
{
    for(j=kink-2;j<NB;j++)
    {
        polymer_bef[j]=polymer[j];
    }
}
else
{
    for(j=kink-1;j<NB;j++)
    {
        polymer_bef[j]=polymer[j];
    }
}

for(j=kink-1;j<=kink+2;j++)
{
    if((j>=1)&&(j<NB-1))
    {
        T_bef[j][1][1]=T[j][1][1];
        T_bef[j][1][2]=T[j][1][2];
        T_bef[j][1][3]=T[j][1][3];
        T_bef[j][2][1]=T[j][2][1];
        T_bef[j][2][2]=T[j][2][2];
        T_bef[j][2][3]=T[j][2][3];
        T_bef[j][3][1]=T[j][3][1];
        T_bef[j][3][2]=T[j][3][2];
        T_bef[j][3][3]=T[j][3][3];
    }
}

for(j=kink-1;j<NB;j++)
{
    if(j>0)
    {
        TT_bef[j][1][1]=TT[j][1][1];
        TT_bef[j][1][2]=TT[j][1][2];
        TT_bef[j][1][3]=TT[j][1][3];
        TT_bef[j][2][1]=TT[j][2][1];
        TT_bef[j][2][2]=TT[j][2][2];
        TT_bef[j][2][3]=TT[j][2][3];
        TT_bef[j][3][1]=TT[j][3][1];
        TT_bef[j][3][2]=TT[j][3][2];
    }
TT_bef[j][3][3]=TT[j][3][3];
}
}

/*Generation of axis to pivot kink around.*/
line.del_x=polymer[kink+1].x-polymer[kink-1].x;
line.del_y=polymer[kink+1].y-polymer[kink-1].y;
line.del_z=polymer[kink+1].z-polymer[kink-1].z;

mag1=sqrt((line.del_x*line.del_x)+(line.del_y*line.del_y)+(line.del_z*line.del_z));
line.n1=line.del_x/mag1;
line.n2=line.del_y/mag1;
line.n3=line.del_z/mag1;

line.x=polymer[kink].x-polymer[kink-1].x;
line.y=polymer[kink].y-polymer[kink-1].y;
line.z=polymer[kink].z-polymer[kink-1].z;
mag2=sqrt((line.x*line.x)+(line.y*line.y)+(line.z*line.z));

/* r cross n*/
line.rn1=line.y*line.n3-line.z*line.n2;
line.rn2=line.z*line.n1-line.x*line.n3;
line.rn3=line.x*line.n2-line.y*line.n1;

/* n(n dot r)*/
ndotr=line.x*line.n1+line.y*line.n2+line.z*line.n3;
line.nr1=line.n1*ndotr;
line.nr2=line.n2*ndotr;
line.nr3=line.n3*ndotr;

/* r_new = n(n dot r)+[r- n(n dot r)]cos(delta)+(r cross n)sin(delta)/
line.xnew=line.nr1+(line.x-line.nr1)*AV[delta].cosine+line.rn1*AV[delta].sine;
line.ynew=line.nr2+(line.y-line.nr2)*AV[delta].cosine+line.rn2*AV[delta].sine;
line.znew=line.nr3+(line.z-line.nr3)*AV[delta].cosine+line.rn3*AV[delta].sine;

/* printf("cos = %f sin = %f   %f
", cos(delta), sin(delta), mag1);*/

polymer[kink].x = line.xnew+polymer[kink-1].x;
polymer[kink].y = line.ynew+polymer[kink-1].y;
polymer[kink].z = line.znew+polymer[kink-1].z;

/*****************************/
/*Recalc of Bond Angles.*/
/*****************************/
for(i=kink-1;i<=kink+2;i++)
{
    /*Calculation of bond(theta) angle.*/
    if(i<kink+2)
    {
        if (i>0 & i<(NB-1))
        {
            line.del_x=polymer[i+1].x-polymer[i].x;
            line.del_y=polymer[i+1].y-polymer[i].y;
            line.del_z=polymer[i+1].z-polymer[i].z;

            mag1=sqrt((line.del_x*line.del_x)+(line.del_y*line.del_y)+(line.del_z*line.del_z));

            line.x=polymer[i].x-polymer[i-1].x;
            line.y=polymer[i].y-polymer[i-1].y;
            line.z=polymer[i].z-polymer[i-1].z;

            mag2=sqrt((line.x*line.x)+(line.y*line.y)+(line.z*line.z));
            polymer[i].the=acos(-(line.x*line.del_x+line.y*line.del_y+line.z*line.del_z)/(mag1*mag2));
        }
    }
}

/*Calculation of dihedral(phi) angle.*/
if (i>1 & i<(NB-1))
{
    x=polymer[i-1].x-polymer[i-2].x;
    y=polymer[i-1].y-polymer[i-2].y;
    z=polymer[i-1].z-polymer[i-2].z;

    /*x2=line.x;
    y2=line.y;
    z2=line.z;*/
    x2=polymer[i].x-polymer[i-1].x;
    y2=polymer[i].y-polymer[i-1].y;
    z2=polymer[i].z-polymer[i-1].z;

    /*x3=line.del_x;
    y3=line.del_y;
    z3=line.del_z;*/
    x3=polymer[i+1].x-polymer[i].x;
    y3=polymer[i+1].y-polymer[i].y;
    z3=polymer[i+1].z-polymer[i].z;

    a1=y*z2-y2*z;
\[a2 = x2*z-x*z2;\]
\[a3 = x*y2-x2*y;\]
\[c1 = y2*z3-y3*z2;\]
\[c2 = x3*z2-x2*z3;\]
\[c3 = x2*y3-x3*y2;\]
\[d1 = a2*c3-c2*a3;\]
\[d2 = -a1*c3+c1*a3;\]
\[d3 = a1*c2-c1*a2;\]

/*angle between the vectors.*/

\[
polymer[i].phi = \arccos((a1*c1+a2*c2+a3*c3)/(\sqrt{(a1*a1+a2*a2+a3*a3) * (c1*c1+c2*c2+c3*c3)}));
\]

if((x2*d1+y2*d2+z2*d3)<0)
  polymer[i].phi = 2.0*PI-polymer[i].phi;
}
}

/* Computation of the new T matrices */

for(i=kink-1;i<=kink+2;i++)
{
  if (i>1 & i<(NB-1))
  {
    AV_theta_cosine = cos(polymer[i].the);
    AV_theta_sine = sin(polymer[i].the);
    AV_phi_cosine = cos(polymer[i].phi);
    AV_phi_sine = sin(polymer[i].phi);
    T[i][1][1] = -AV_theta_cosine;
    T[i][1][2] = AV_theta_sine;
    T[i][1][3] = 0.0;
    T[i][2][1] = -AV_theta_sine*AV_phi_cosine;
    T[i][2][2] = -AV_theta_cosine*AV_phi_cosine;
    T[i][2][3] = -AV_phi_sine;
    T[i][3][1] = -AV_theta_sine*AV_phi_sine;
    T[i][3][2] = -AV_theta_cosine*AV_phi_sine;
    T[i][3][3] = AV_phi_cosine;
  }
}

for(i=kink-1;i<NB;i++)
{
  if (i>0 & i<NB)
  {
    
  }
}
void Move_rotate()
{
    int subscript;
    int arg1, arg2;
    double AV_theta_cosine, AV_theta_sine, AV_phi_cosine, AV_phi_sine;
    register int i,j;
    double sinBA,cosBA,sinPhi,cosPhi,auxn;

    pivot=(int)(2+genrand()*(NB-2));

    T_bef[pivot][1][1]=T[pivot][1][1];
    T_bef[pivot][1][2]=T[pivot][1][2];
    T_bef[pivot][1][3]=T[pivot][1][3];
    T_bef[pivot][2][1]=T[pivot][2][1];
    T_bef[pivot][2][2]=T[pivot][2][2];
    T_bef[pivot][2][3]=T[pivot][2][3];
    T_bef[pivot][3][1]=T[pivot][3][1];
    T_bef[pivot][3][2]=T[pivot][3][2];
    T_bef[pivot][3][3]=T[pivot][3][3];
    polymer[pivot].phi=2*PI*genrand();

    AV_theta_cosine=cos(polymer[pivot].the);
    AV_theta_sine=sin(polymer[pivot].the);
    AV_phi_cosine=cos(polymer[pivot].phi);
    AV_phi_sine=sin(polymer[pivot].phi);
    T[pivot][1][1]=-AV_theta_cosine;
    T[pivot][1][2]=AV_theta_sine;
    T[pivot][1][3]=0.0;
\[ T[pivot][2][1]=-AV_{\theta \sin} AV_{\phi \cos}; \]
\[ T[pivot][2][2]=-AV_{\theta \cos} AV_{\phi \cos}; \]
\[ T[pivot][2][3]=-AV_{\phi \sin}; \]
\[ T[pivot][3][1]=-AV_{\theta \sin} AV_{\phi \sin}; \]
\[ T[pivot][3][2]=-AV_{\theta \cos} AV_{\phi \sin}; \]
\[ T[pivot][3][3]=AV_{\phi \cos}; \]

for(j=pivot; j<NB; j++)
{
    TT_bef[j][1][1]=TT[j][1][1];
    TT_bef[j][1][2]=TT[j][1][2];
    TT_bef[j][1][3]=TT[j][1][3];
    TT_bef[j][2][1]=TT[j][2][1];
    TT_bef[j][2][2]=TT[j][2][2];
    TT_bef[j][2][3]=TT[j][2][3];
    TT_bef[j][3][1]=TT[j][3][1];
    TT_bef[j][3][2]=TT[j][3][2];
    TT_bef[j][3][3]=TT[j][3][3];
    MatrixP(j);
}

for(j=pivot+1; j<NB; j++)
{
    polymer_bef[j]=polymer[j];
    polymer[j].x= polymer[j-1].x+TT[j-1][1][1]*BL;
    polymer[j].y= polymer[j-1].y+TT[j-1][2][1]*BL;
    polymer[j].z= polymer[j-1].z+TT[j-1][3][1]*BL;
}

double min(double a, double b)
{
    if (a<b) return a; else return b;
}

int RWA(int a, int b)
{
    long int rRWA=1;
    double temprandom;
    double ratio;
    double prob;
    ratio = exp(a-b);
    prob = min(1,ratio);
    temprandom = genrand();
if (temprandom<prob) rRWA=0;
return rRWA;
}

/********************MAIN FUNCTION***********************************/

int main()
{
  int gen_count=0;
  int pivot_count=0;
  int kink_count=0;
  int DOScheck=0;
  int hiscount=0;
  int histprint=0;
  long int kounter=0;
  long int kounterb=0;
  long int kouterc=0;
  int i,k;
  int j=0;
  int EPrev=0;
  int kinkmin=NB;
  int kinkmax=0;
  int ktrack=0;
  int know=0;
  double keng;
  int HBtrack=0;
  int FLAG=0;
  double binsum;
  int nil=0;
  int flat=0;
  int line=0;
  char
  filename1[30],filename2[30],filename3[30],filename4[30],filename5[30],filename6[30],f
  ilename7[30],filename8[30];
  double f=e;
  double binavg;
  double test;
double hist_tol;
double EtoE_old=0;

sprintf(filename1,"0DOS20_10.txt");
if ((DOS=fopen(filename1,"w"))==NULL)
{
    printf("Error writing DOS file \n");
    exit(1);
}

fprintf(DOS,"DENSITY OF STATES ARRAYS\n\n");

if (fclose(DOS)!=0)
{
    printf("Error in DOS\n");
    exit(1);
}

sprintf(filename2,"0pre_sim_data20_10.txt");
if ((pre=fopen(filename2,"w"))==NULL)
{
    printf("Error writing trial file \n");
    exit(1);
}

sprintf(filename3,"0acoordinates20_10");
if ((acoord=fopen(filename3,"w"))==NULL)
{
    printf("Error writing trial file \n");
    exit(1);
}

sprintf(filename4,"0bcoordinates20_10");
if ((bcoord=fopen(filename4,"w"))==NULL)
{
    printf("Error writing trial file \n");
    exit(1);
}

sprintf(filename5,"0ccoordinates20_10");
if ((ccoord=fopen(filename5,"w"))==NULL)
{
    printf("Error writing trial file \n");
}
exit(1);
}

sprintf(filename6,"0extra_coor20_10");
if (extra=fopen(filename6,"w")==NULL)
{
    printf("Error writing trial file \
");
    exit(1);
}

sprintf(filename7,"0extra_hbonds20_10");
if (extra_1=fopen(filename7,"w")==NULL)
{
    printf("Error writing trial file \
");
    exit(1);
}

sprintf(filename8,"0extra_hist20_10");
if (extra_2=fopen(filename8,"w")==NULL)
{
    printf("Error writing trial file \
");
    exit(1);
}

/* Calculation of square root array*/
for (i=0; i<sqrtsize; i++)
{
    Asqrt[i]=sqrt(((double)(i))/1000.0);
}

for (i=0;i<36000000;i++)
{
    AV[i].cosine=cos(((i/100000)*PI)/180);
    AV[i].sine=sin(((i/100000)*PI)/180);
}

/* Initialize Arrays */
for(i=0;i<=NB;i++)
{
    totHB[i]=0;
}

for(i=0;i<=NB;i++)
{
    for(j=0;j<kbin;j++)
{ /* DOSarray[i][j]=0; */
    hist[i][j]=0;
    hist_extra[i][j]=0;
}
}

for(i=0;i<=NB;i++)
{
    if(i==0)
    {
        for(j=4;j<16;j++)
        {
            check[i][j]=1;
        }
    }
    else if(i==2)
    {
        for(j=6;j<16;j++)
        {
            check[i][j]=1;
        }
    }
    else if(i==4)
    {
        /* for(j=6;j<14;j++) */
        for(j=6;j<13;j++)
        {
            check[i][j]=1;
        }
    }
    else
    {
        check[i][j]=0;
    }
}

/* First generate a good polymer chain configuration.*/
do
{
    ChainGenFRM();
    keng=Energy();
    EPrev=(int)(keng);
    gen_count++;
}
while (EPrev==1000);

HBtrack=hbonds;
ktrack=(int)(keng);

/* Implementing moves*/

fprintf(acoord,"A Coordinates\n");
fprintf(bcoord,"B Coordinates\n");
fprintf(ccoord,"C Coordinates\n");

do{
    /*kounter++;
    /* Entering kink */
    for(k=1;k<10;k++)
    {
        do{
            kink_count++;
            kink_move();

            if(kinkmin<kink)
                kinkmin=kink;
            if(kinkmax<kink)
                kinkmax=kink;

            keng=Energy();
            EPrev=(int)(keng);

            if(EPrev!=1000)
            {
                know=(int)(keng);
                FLAG=RWA(DOSarray[HBtrack][ktrack],DOSarray[hbonds][know]);

                if(FLAG==0)
                {
                    if(check[hbonds][know]==0)
                    {
                        hist_extra[hbonds][know]=hist_extra[hbonds][know]+1.0;
                        if(hist_extra[hbonds][know]<1000)
                        {
                            for(i=0;i<NB;i++)
                            {
                                fprintf(extra_1,"H-Bonds ");
                                for(i=0;i<NB;i++)
                                {
                                    fprintf(extra_1,"H-Bonds ");
                                }
                            }
                        }
                    }
                }
            }
        }
    }
}
while (EPrev==1000);
if (polymer[i].HB!=0)
{
    fprintf(extra_1,"%d %d    ",i+1,polymer[i].HB+1);
}
fprintf(extra_1,"\n");
for(i=0;i<NB;i++)
    if (polymer[i].HB!=0)
        
        fprintf(extra_1,"Dihedral[%d]= %f ",i+1,180*polymer[i].phi/PI);
    fprintf(extra_1,"\n");

line=0;
fprintf(extra,"%8.2f", polymer[i].x);
line++;
if(line%NB==0)
    fprintf(extra,"\n");

fprintf(acoord,"%8.2f", polymer[i].y);
line++;
if(line%NB==0)
    fprintf(extra,"\n");

fprintf(extra,"%8.2f", polymer[i].z);
line++;
if(line%NB==0)
    fprintf(extra,"\n");

if (i==NB-1)
{ 
    fprintf(extra,"  100.00  ");
    fprintf(extra," 100.00  ");
    fprintf(extra," 100.00\n");
}
}
else
{
    hist[hbonds][know]=hist[hbonds][know]+1.0;
    DOSarray[hbonds][know]=DOSarray[hbonds][know]+log(f);
    HBtrack=hbonds;
    ktrack=know;
}
else
{
    hist[HBtrack][ktrack]=hist[HBtrack][ktrack]+1.0;
    DOSarray[HBtrack][ktrack]=DOSarray[HBtrack][ktrack]+log(f);
}

hiscount++;

if(EPrev==1000 || FLAG==1)
{
    if(kink>1)
    {
        for(i=kink-1;i<=kink+2;i++)
            polymer[i]=polymer_bef[i];
        for(i=kink-1;i<NB;i++)
            {
                TT[i][1][1]=TT_bef[i][1][1];
                TT[i][1][2]=TT_bef[i][1][2];
                TT[i][1][3]=TT_bef[i][1][3];
                TT[i][2][1]=TT_bef[i][2][1];
                TT[i][2][2]=TT_bef[i][2][2];
                TT[i][2][3]=TT_bef[i][2][3];
                TT[i][3][1]=TT_bef[i][3][1];
                TT[i][3][2]=TT_bef[i][3][2];
                TT[i][3][3]=TT_bef[i][3][3];
            }
        for(i=kink-1;i<=kink+2;i++)
            {
                T[i][1][1]=T_bef[i][1][1];
                T[i][1][2]=T_bef[i][1][2];
                T[i][1][3]=T_bef[i][1][3];
                T[i][2][1]=T_bef[i][2][1];
                T[i][2][2]=T_bef[i][2][2];
                T[i][2][3]=T_bef[i][2][3];
                T[i][3][1]=T_bef[i][3][1];
                T[i][3][2]=T_bef[i][3][2];
                T[i][3][3]=T_bef[i][3][3];
            }
    }

    else
    {
        for(i=kink;i<NB;i++)
            polymer[i]=polymer_bef[i];
        for(i=kink;i<NB;i++)
            {
{ 
    TT[i][1][1]=TT_bef[i][1][1]; 
    TT[i][1][2]=TT_bef[i][1][2]; 
    TT[i][1][3]=TT_bef[i][1][3]; 
    TT[i][2][1]=TT_bef[i][2][1]; 
    TT[i][2][2]=TT_bef[i][2][2]; 
    TT[i][2][3]=TT_bef[i][2][3]; 
    TT[i][3][1]=TT_bef[i][3][1]; 
    TT[i][3][2]=TT_bef[i][3][2]; 
    TT[i][3][3]=TT_bef[i][3][3]; 
}
for(i=kink;i<=kink+2;i++)
{
    T[i][1][1]=T_bef[i][1][1]; 
    T[i][1][2]=T_bef[i][1][2]; 
    T[i][1][3]=T_bef[i][1][3]; 
    T[i][2][1]=T_bef[i][2][1]; 
    T[i][2][2]=T_bef[i][2][2]; 
    T[i][2][3]=T_bef[i][2][3]; 
    T[i][3][1]=T_bef[i][3][1]; 
    T[i][3][2]=T_bef[i][3][2]; 
    T[i][3][3]=T_bef[i][3][3]; 
}
}
while(EPrev==1000); 
} 

/* Entering Pivot */
do{
    pivot_count++; 
    Move_rotate();
    keng=Energy();
    EPrev=(int)(keng);
    if(EPrev!=1000)
    {
        if(fabs(EtoE-EtoE_old)>0.01)
        {
            EtoE_old=EtoE;
            if(hbonds>=4)
if(hbonds==4)
{
  kountera++;
  if(kountera<10000)
  {
    printf("4 H-bonds %d ",kountera);
    Energy1();
    line=0;
    for(i=0;i<NB;i++)
    {
      fprintf(acoord,"%8.2f", polymer[i].x);
      line++;
      if(line%10==0)
        fprintf(acoord,"\n");

      fprintf(acoord,"%8.2f", polymer[i].y);
      line++;
      if(line%10==0)
        fprintf(acoord,"\n");

      fprintf(acoord,"%8.2f", polymer[i].z);
      line++;
      if(line%10==0)
        fprintf(acoord,"\n");

      if (i==NB-1)
      {
        fprintf(acoord,"  100.00  ");
        fprintf(acoord,"100.00 ");
        fprintf(acoord,"100.00\n");
      }
    }
  }
}
else if(hbonds==6)
{
  kounterb++;
  if(kounterb<10000)
  {
    printf("6 H-bonds %d ",kounterb);
    Energy1();
    line=0;
    for(i=0;i<NB;i++)
    {
      fprintf(acoord,"%8.2f", polymer[i].x);
      line++;
      if(line%10==0)
        fprintf(acoord,"\n");

      fprintf(acoord,"%8.2f", polymer[i].y);
      line++;
      if(line%10==0)
        fprintf(acoord,"\n");

      fprintf(acoord,"%8.2f", polymer[i].z);
      line++;
      if(line%10==0)
        fprintf(acoord,"\n");

      if (i==NB-1)
      {
        fprintf(acoord,"  100.00  ");
        fprintf(acoord,"100.00 ");
        fprintf(acoord,"100.00\n");
      }
    }
  }
}
fprintf(bcoord,"%8.2f", polymer[i].x);
line++;
if(line%10==0)
    fprintf(bcoord,"\n");

fprintf(bcoord,"%8.2f", polymer[i].y);
line++;
if(line%10==0)
    fprintf(bcoord,"\n");

fprintf(bcoord,"%8.2f", polymer[i].z);
line++;
if(line%10==0)
    fprintf(bcoord,"\n");

if (i==NB-1)
{
    fprintf(bcoord," 100.00  ");
    fprintf(bcoord,"100.00  ");
    fprintf(bcoord,"100.00
");
}
}
}
}
}
else if(hbonds==8)
{
    kounterc++;
    if(kounterc<10000)
    {
        printf("8 H-bonds %d ",kounterc);
        Energy1();
        line=0;
        for(i=0;i<NB;i++)
        {
            fprintf(ccoord,"%8.2f", polymer[i].x);
            line++;
            if(line%10==0)
                fprintf(ccoord,"\n");

            fprintf(ccoord,"%8.2f", polymer[i].y);
            line++;
            if(line%10==0)
                fprintf(ccoord,"\n");

            fprintf(ccoord,"%8.2f", polymer[i].z);

    printf("8 H-bonds %d ",kounterc);
    Energy1();
    line=0;
    for(i=0;i<NB;i++)
    {
        fprintf(ccoord,"%8.2f", polymer[i].x);
        line++;
        if(line%10==0)
            fprintf(ccoord,"\n");

        fprintf(ccoord,"%8.2f", polymer[i].y);
        line++;
        if(line%10==0)
            fprintf(ccoord,"\n");

        fprintf(ccoord,"%8.2f", polymer[i].z);

    printf("8 H-bonds %d ",kounterc);
    Energy1();
    line=0;
    for(i=0;i<NB;i++)
    {
        fprintf(ccoord,"%8.2f", polymer[i].x);
        line++;
        if(line%10==0)
            fprintf(ccoord,"\n");

        fprintf(ccoord,"%8.2f", polymer[i].y);
        line++;
        if(line%10==0)
            fprintf(ccoord,"\n");

        fprintf(ccoord,"%8.2f", polymer[i].z);


line++;  
if(line%10==0)  
    fprintf(ccoord,"\n");
if (i==NB-1)  
{  
    fprintf(ccoord,"  100.00 ");  
    fprintf(ccoord,"100.00 ");  
    fprintf(ccoord,"100.00\n");  
}
else  
    printf("ODD H-bonds= %d\n",hbonds);
}
/*****************************/

know=(int)(keng);  
FLAG=RWA(DOSarray[HBtrack][ktrack],DOSarray[hbonds][know]);  
if(FLAG==0)  
{  
    if(check[hbonds][know]==0)  
    {  
        hist_extra[hbonds][know]=hist_extra[hbonds][know]+1.0;  
        
        if(hist_extra[hbonds][know]<1000)  
        {  
            fprintf(extra_1,"H-Bonds ");  
            for(i=0;i<NB;i++)  
                if (polymer[i].HB!=0)  
                {  
                    fprintf(extra_1,"%d %d ",i+1,polymer[i].HB+1);  
                }  
            fprintf(extra_1,\n");  
            for(i=0;i<NB;i++)  
                if (polymer[i].HB!=0)  
                {  
                    fprintf(extra_1,"Dihedral[%d]= %f ",i+1,180*polymer[i].phi/PI);  
                }  
            fprintf(extra_1,\n");  
        }  
    }  
}
line=0;
for(i=0;i<NB;i++)
{
    fprintf(extra,"%8.2f", polymer[i].x);
    line++;
    if(line%NB==0)
        fprintf(extra,"
");

    fprintf(acoord,"%8.2f", polymer[i].y);
    line++;
    if(line%NB==0)
        fprintf(extra,"\n");

    fprintf(extra,"%8.2f", polymer[i].z);
    line++;
    if(line%NB==0)
        fprintf(extra,"\n");

    if (i==NB-1)
    {
        fprintf(extra,"  100.00  ");
        fprintf(extra,"100.00  ");
        fprintf(extra,"100.00\n");
    }
}
else
{
    hist[hbonds][know]=hist[hbonds][know]+1.0;
    DOSarray[hbonds][know]=DOSarray[hbonds][know]+log(f);
    HBtrack=hbonds;
    ktrack=know;
}
else
{

    hist[HBtrack][ktrack]=hist[HBtrack][ktrack]+1.0;
    DOSarray[HBtrack][ktrack]=DOSarray[HBtrack][ktrack]+log(f);
}

hiscount++;
}

if(EPrev==1000 || FLAG==1)
for(i=pivot+1;i<NB;i++)
{
    polymer[i]=polymer_bef[i];
}

for(i=pivot;i<NB;i++)
{
    TT[i][1][1]=TT_bef[i][1][1];
    TT[i][1][2]=TT_bef[i][1][2];
    TT[i][1][3]=TT_bef[i][1][3];
    TT[i][2][1]=TT_bef[i][2][1];
    TT[i][2][2]=TT_bef[i][2][2];
    TT[i][2][3]=TT_bef[i][2][3];
    TT[i][3][1]=TT_bef[i][3][1];
    TT[i][3][2]=TT_bef[i][3][2];
    TT[i][3][3]=TT_bef[i][3][3];

    T[pivot][1][1]=T_bef[pivot][1][1];
    T[pivot][1][2]=T_bef[pivot][1][2];
    T[pivot][1][3]=T_bef[pivot][1][3];
    T[pivot][2][1]=T_bef[pivot][2][1];
    T[pivot][2][2]=T_bef[pivot][2][2];
    T[pivot][2][3]=T_bef[pivot][2][3];
    T[pivot][3][1]=T_bef[pivot][3][1];
    T[pivot][3][2]=T_bef[pivot][3][2];
    T[pivot][3][3]=T_bef[pivot][3][3];
}
} while(EPrev==1000);

if(hiscount>100000)
{
    hiscount=0;
    histprint++;
if(histprint>100)
{
  /* histprint=0; */
  for(i=0;i<=NB;i++)
  {
    for(j=0;j<kbin;j++)
    {
      fprintf(pre,"%f ",hist[i][j]);
    }
    fprintf(pre,"
");
  }
}

nil=29;
binsum=0;
flat=0;
for(i=0;i<=NB;i++)
{
  for(j=0;j<kbin;j++)
  {
    if(check[i][j]==1)
    {
      binsum=binsum+hist[i][j];
    }
  }
}

binavg=binsum,nil;
hist_tol=0.2*binavg;

for(i=0;i<=NB;i++)
{
  for(j=0;j<kbin;j++)
  {
    test=fabs(binavg-hist[i][j]);
    if(test<hist_tol)
    {
      flat++;
    }
  }
}

if(histprint>100)
{
  histprint=0;
  fprintf(pre,"Flat bins= %d  Bin avg= %f\n",flat,binavg);
}
if(flat==nil)
{
    for(i=0;i<=NB;i++)
    {
        for(j=0;j<kbin;j++)
        {
            hist[i][j]=0;
        }
    }
}

/******************************/
/**** Printing DOS to file ****/
/*****************************/
sprintf(filename1,"0DOS20_10.txt");
if ((DOS=fopen(filename1,"a"))==NULL)
{
    printf("Error writing DOS file \n");
    exit(1);
}
fprintf(DOS,"\n\n\n");
for(i=0;i<=NB;i++)
{
    for(j=0;j<kbin;j++)
    {
        fprintf(DOS,"%8.2f ",DOSarray[i][j]);
    }
    fprintf(DOS,"\n");
}
fprintf(DOS,"f= %f\n\n\n",f);
if (fclose(DOS)!=0)
{
    printf("Error in DOS");
    exit(1);
}

sprintf(filename8,"0extra_hist20_10");
if ((extra_2=fopen(filename8,"a"))==NULL)
{
    printf("Error writing DOS file \n");
    exit(1);
}
fprintf(extra_2,"\n\n");
for(i=0;i<=NB;i++)
{
    for(j=0;j<kbin;j++)
    {
        fprintf(extra_2,"%f ",hist_extra[i][j]);
    }
    fprintf(extra_2,"\n");
}
fprintf(extra_2,"f=%f\n",f);
fprintf(extra_2,"\n\n");
for(i=0;i<=NB;i++)
{
    fprintf(extra_2,"Total Conformations with %d H-bonded beads=\n",i,totHB[i]);
}
if (fclose(extra_2)!=0)
{
    printf("Error in ex2");
    exit(1);
}
/******************************/
/**** Printing DOS to file ****/
/******************************/

f=sqrt(f);
}
}
}while(f>1.0000001);

/**************************************************************************/
**************************************************************************/
**************************************************************************/

fprintf(extra_1,"Min kink energy= %f\n",kmin);
fprintf(extra_1,"Max kink energy= %f\n",kmax);
sprintf(filename8,"0extra_hist20_10");
if ((extra_2=fopen(filename8,"a"))==NULL)
{
    printf("Error writing DOS file\n");
    exit(1);
}

fprintf(extra_2,"\n\n");
for(i=0;i<=NB;i++)
{
    for(j=0;j<kbin;j++)
    {
        fprintf(extra_2,"%f ",hist_extra[i][j]);
    }
    fprintf(extra_2,"\n");
}
fprintf(extra_2,"\n\n");
for(i=0;i<=NB;i++)
{
    fprintf(extra_2,"Total Conformations with %d H-bonded beads= %d\n",i,totHB[i]);
}

if (fclose(extra_2)!=0)
{
    printf("Error in ex2");
    exit(1);
}

if (fclose(pre)!=0)
{
    printf("Error in pre");
    exit(1);
}

if (fclose(acoord)!=0)
{
    printf("Error in coord");
    exit(1);
}

if (fclose(bcoord)!=0)
{
    printf("Error in coord");
    exit(1);
if (fclose(ccoord)!=0)
{
    printf("Error in coord");
    exit(1);
}

if (fclose(extra)!=0)
{
    printf("Error in e");
    exit(1);
}

if (fclose(extra_1)!=0)
{
    printf("Error in e1");
    exit(1);
}

return 0;
APPENDIX C
DATA ANALYSIS PROGRAM

/* Standard Libraries */
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>
#include <ctype.h>
#include <time.h>

/* Constants */
#define C -650                  /* Smaller gives higher peak, 650 gives peak at 300 degrees */
#define NB 10
#define kbin 20
#define amax 1000

/* Arrays */
double DOS[NB+1][kbin];       /* Density of states read in from file */
double E[NB+1][kbin];         /* */
double z_sum[NB+1][kbin];     /* */
double Eng1[601][amax];       /* <E^2> */
double Eng2[601][amax];       /* <E>^2 */
double Cv[601][amax];         /* Heat Capacity */
double Z_array[601][amax];    /* Partition function */
double F_array[601][amax];    /* Free Energy */
double U_array[601][amax];    /* Internal Energy */
double S_array[601][amax];    /* Entropy */


int main()
{
    char filename1[30],filename2[30],filename3[30],filename4[30];
    char filename5[30],filename6[30],filename7[30],filename8[30],w[30];
    int i,j,k;
    int T,A;
    double T1;
double alpha;
double zsum=0;
double E1sum=0, E2sum=0;
double DOSsum=0;

sprintf(filename1,"DOS_analysis.txt");
if ((array_1=fopen(filename1,"w"))==NULL)
{
    printf("Error writing DOS file \n");
    exit(1);
}

sprintf(filename2,"Z_array.txt");
if ((Z=fopen(filename2,"w"))==NULL)
{
    printf("Error writing Z_array file \n");
    exit(1);
}

sprintf(filename3,"F_array.txt");
if ((F=fopen(filename3,"w"))==NULL)
{
    printf("Error writing F_array file \n");
    exit(1);
}

sprintf(filename4,"U_array.txt");
if ((U=fopen(filename4,"w"))==NULL)
{
    printf("Error writing U_array file \n");
    exit(1);
}

sprintf(filename5,"S_array.txt");
if ((S=fopen(filename5,"w"))==NULL)
{
    printf("Error writing S_array file \n");
    exit(1);
}

sprintf(filename6,"Cv_array.txt");
if ((Cva=fopen(filename6,"w"))==NULL)
{
    printf("Error writing Cv_array file \n");
    exit(1);
}
sprintf(filename7, "DOS_array.txt");
if ((DOSa=fopen(filename7, "w"))==NULL)
{
    printf("Error writing DOS_array file \n");
    exit(1);
}

sprintf(filename8, "Cv_anal.txt");
if ((Cvb=fopen(filename8, "w"))==NULL)
{
    printf("Error writing Cv file \n");
    exit(1);
}

if (!(input_1=fopen("input1.txt", "r"))==NULL)
{
    printf("Error reading file \n");
    exit(1);
}

/* Initialize Arrays */
for(i=0;i<NB+1;i++)
{
    for(j=0;j<kbin;j++)
    {
        E[i][j]=0;
        z_sum[i][j]=0;
        fscanf(input_1, "%s ",w);
        DOS[i][j]=atof(w);
    }
}

for(T=0; T<601; T++)
{
    for(A=0; A<amax; A++)
    {
        Z_array[T][A]=0;
        F_array[T][A]=0;
        U_array[T][A]=0;
        S_array[T][A]=0;
        Eng1[T][A]=0;
        Eng2[T][A]=0;
        Cv[T][A]=0;
    }
}
for(T=25;T<626;T++)
{
    for(A=0;A<amax;A++)
    {
        T1=(double)(T);
        zsum=0;
        DOSsum=0;
        E1sum=0;
        E2sum=0;
        alpha=(A+1)/1;

        for(i=0;i<NB+1;i++)
        {
            for(j=0;j<kbin;j++)
            {
                E[i][j]=((i/2)*C+j*alpha);
                z_sum[i][j]=DOS[i][j]*exp(-(E[i][j])/T1);
                zsum=zsum+z_sum[i][j];
                E1sum=E1sum+E[i][j]*E[i][j]*z_sum[i][j];
                E2sum=E2sum+E[i][j]*z_sum[i][j];
            }
        }
        Z_array[T-25][A]=zsum;
        F_array[T-25][A]=-T1*log(Z_array[T-25][A]);
        Eng1[T-25][A]=E1sum/zsum;
        Eng2[T-25][A]=(E2sum/zsum)*(E2sum/zsum);
        U_array[T-25][A]=(E2sum/zsum);
        S_array[T-25][A]=(U_array[T-25][A]-F_array[T-25][A])/T1;
        Cv[T-25][A]=(Eng1[T-25][A]-Eng2[T-25][A])/(T1*T1);
    }
}

/* Printing Z */
/*fprintf(array_1,"\n\n\nZ array\n\n");

for(T=0;T<601;T++)
{
    for(A=0;A<amax;A++)
    {
        fprintf(array_1,"%8.2f ",Z_array[T][A]);
    }
}
fprintf(array_1,"
");
} /*

for(T=0;T<601;T=T+10)
{
    for(A=0;A<amax;A=A+10)
    {
        fprintf(Z,"%d %d %f\n",T+25,A+1,Z_array[T][A]);
    }
}

/* Printing F */
/*fprintf(array_1,"\n\n\nF array\n\n");
for(T=0;T<601;T++)
{
    for(A=0;A<amax;A++)
    {
        fprintf(array_1,"%8.2f ",F_array[T][A]);
    }
    fprintf(array_1,"\n");
} */

for(T=0;T<601;T=T+10)
{
    for(A=0;A<amax;A=A+10)
    {
        fprintf(F,"%d %d %f\n",T+25,A+1,F_array[T][A]);
    }
}

/* Printing U */
/*fprintf(array_1,"\n\n\nU array\n\n");
for(T=0;T<601;T++)
{
    for(A=0;A<amax;A++)
    {
        fprintf(array_1,"%8.3f ",U_array[T][A]);
    }
    fprintf(array_1,"\n");
} */

for(T=0;T<601;T=T+10)
{
    for(A=0;A<amax;A=A+10)
    {
    }
fprintf(U,"%d %d %f\n",T+25,A+1,U_array[T][A]);
}
}

/* Printing S */
/*fprintf(array_1,"\n\n\nS array\n\n");*/
for(T=0;T<601;T++)
{
   for(A=0;A<amax;A++)
   {
      fprintf(array_1,"%8.5f ",S_array[T][A]);
   }
   fprintf(array_1,"\n");
}/*

for(T=0;T<601;T=T+10)
{
   for(A=0;A<amax;A=A+10)
   {
      fprintf(S,"%d %d %f\n",T+25,A+1,S_array[T][A]);
   }
}
/* Printing Cv */
/*fprintf(array_1,"\n\n\nCv array\n\n");*/
for(T=0;T<601;T++)
{
   for(A=0;A<amax;A++)
   {
      fprintf(array_1,"%8.5f ",Cv[T][A]);
   }
   fprintf(array_1,"\n");
}/*

for(T=0;T<601;T++)
{
   for(A=0;A<amax;A++)
   {
      fprintf(Cvb,"%8.5f ",Cvb[T][A]);
   }
   fprintf(Cvb,"\n");
}
for(T=0;T<601;T=T+10)
{
for(A=0;A<amax;A=A+10)  
  {  
    fprintf(Cva, "%d %d %f\n", T+25, A+1, Cv[T][A]);  
  }  

/* Printing DOS */  
fprintf(array_1, "\n\nDOS array\n\n");  
for(i=0;i<NB+1;i++)  
  {  
    for(j=0;j<kbin;j++)  
      {  
        fprintf(array_1, "%8.2f ", DOS[i][j]);  
      }  
    fprintf(array_1, "\n");  
  }  
for(i=0;i<NB+1;i++)  
  {  
    for(j=0;j<kbin;j++)  
      {  
        fprintf(DOSa, "%d %d %f\n", i, j, DOS[i][j]);  
      }  
  }  

fprintf(array_1, "\n\nE array\n\n");  
for(i=0;i<NB+1;i++)  
  {  
    for(j=0;j<kbin;j++)  
      {  
        fprintf(array_1, "%5.0f ", E[i][j]);  
      }  
    fprintf(array_1, "\n");  
  }  

if (fclose(array_1)!=0)  
  {  
    printf("Error in array");  
    exit(1);  
  }  
if (fclose(Z)!=0)  
  {  
    printf("Error in z");  
  }
exit(1);
}

if (fclose(F)!=0)
{
    printf("Error in f");
    exit(1);
}

if (fclose(U)!=0)
{
    printf("Error in u");
    exit(1);
}

if (fclose(S)!=0)
{
    printf("Error in s");
    exit(1);
}

if (fclose(Cva)!=0)
{
    printf("Error in Cv");
    exit(1);
}

if (fclose(Cvb)!=0)
{
    printf("Error in Cv");
    exit(1);
}

if (fclose(DOSa)!=0)
{
    printf("Error in DOSa");
    exit(1);
}

if (fclose(input_1)!=0)
{
    printf("Error in input");
    exit(1);
}

return 0;
APPENDIX D

PHASE DIAGRAM PROGRAM

/* Standard Libraries */
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>
#include <ctype.h>
#include <time.h>
define amax 1000

double Cv[601][amax];               /* Heat Capacity */

FILE *pd, *input_1;

int main()
{
    int T,A;
    char filename1[30],w[30];

    sprintf(filename1,"Phase_diagram.txt");
    if ((pd=fopen(filename1,"w"))==NULL)
    {
        printf("Error writing Cvb file \n");
        exit(1);
    }

    if ((input_1=fopen("Cv_anal.txt","r"))==NULL)
    {
        printf("Error reading file \n");
        exit(1);
    }

    /* Read in Data */
    for(T=0;T<601;T++)
    {
        for(A=0;A<amax;A++)
        {
            // Code continues here...
        }
    }
}
/* Find maxima in A direction */
fprintf(pd,"A maxima\n");
for(T=0;T<601;T++)
{
    for(A=1;A<amax-1;A++)
    {
        if(Cv[T][A]>Cv[T][A-1])
        {
            if(Cv[T][A]>Cv[T][A+1])
            {
                fprintf(pd,"%d %d %f\n",T+24,A,Cv[T][A]);
            }
        }
    }
}

/* Find maxima in T direction */
fprintf(pd,"\n\nT maxima\n");
for(A=0;A<amax;A++)
{
    for(T=1;T<600;T++)
    {
        if(Cv[T][A]>Cv[T-1][A])
        {
            if(Cv[T][A]>Cv[T+1][A])
            {
                fprintf(pd,"%d %d %f\n",T+24,A,Cv[T][A]);
            }
        }
    }
}
if (fclose(pd)!=0)
{ printf("Error in pd");
    exit(1);
}
if (fclose(input_1)!=0)
{ printf("Error in input");
    exit(1);
}
return 0;