STRUCTURAL ANALYSIS OF THE INTERACTIONS BETWEEN COLLAGEN AND COLLAGEN-BINDING PROTEINS

An Undergraduate Research Scholars Thesis

by

MITCHELL HSU, AARON KIM, and MICHAEL TANG

Submitted to the Undergraduate Research Scholars program at
Texas A&M University
in partial fulfillment of the requirements for the designation as an

UNDERGRADUATE RESEARCH SCHOLAR

Approved by Research Advisor: Dr. Wonmuk Hwang

May 2020

Major: Biomedical Engineering

TABLE OF CONTENTS

ABSTRACT

Structural Analysis of the Interactions Between Collagen and Collagen-binding Proteins

Mitchell Hsu, Aaron Kim, and Michael Tang
Department of Biomedical Engineering
Texas A&M University

Research Advisor: Dr. Wonmuk Hwang
Department of Biomedical Engineering
Texas A&M University

Collagen plays several key roles in cell binding, tissue growth, and structural strengthening of the body. We chose to study three proteins: OSCAR, DDR2, and α-1 integrin. These protein and protein domain models were built through CHARMM scripts by simulations in aqueous, ionized, heated, and energy minimized solutions. These structures were visualized in VMD. Production simulations were then performed on Texas A&M High Performance Research Computing clusters to achieve reasonable statistical trajectories for 10 ns.

During the simulations, DDR2 and human OSCAR were found to be stable structures. α-1 integrin was found to be unstable because it possessed only a single high occupancy hydrogen bond between itself and collagen. During its simulation, we saw signs of dissociation, and suspect that the collagen would have fully dissociated had the simulation continued past 10 ns. This stability problem was emphasized further by the significant difference in the patterning of α-1 integrin's RMSF and B-Factor values. DDR2 and OSCAR had similar patterns between their RMSF and B-Factor values, indicating similar stability to their original coordinates. As such, further analysis was only carried out on DDR2 and OSCAR.

Looking at the distances between collagen strands for DDR2 and OSCAR models, we saw a large distance at the C and N terminals due to the end effect. All of the distances in the central regions of the collagen were roughly 5 Å apart consistently. This indicated that the structures were stable and there were no dissociations between individual collagen strands for DDR2 and OSCAR over the 10 ns simulation.

The torsional angles of triads, explained in the methods, were found for DDR2 and OSCAR. Unwinding behavior was generally found within the segments of collagen that bind to DDR2 and OSCAR. This was identified as a decrease in torsional angle. Further analysis was used to determine the stability of these torsional angles over time. Generally, more hydrogen bonds between each triad and DDR2/OSCAR increased the stability of the torsional angle while unwinding the collagen fibril. Steric stabilization due to protein pockets in DDR2 and OSCAR also aided in torsional angle stabilization.

ACKNOWLEDGMENTS

We would like to thank our research advisor, Dr. Hwang, and our graduate student advisor, Mr. Shi, for their guidance and support throughout the course of this research. We would also like to thank the rest of the Hwang laboratory for their support and the advanced computing resources provided by Texas A&M High Performance Research Computing.

NOMENCLATURE

CBPs	Collagen Binding Proteins
CHARMM	Chemistry at Harvard Macromolecular Mechanics
MMP	Matrix Metalloproteinase
MODELLER	Program for Comparative Protein Structure Modelling by Satisfaction of Spatial Restraint
DDR	Discoidin Domain Receptor
OSCAR	Osteoclast-Associated Immunoglobulin-like Receptor
VMD	Visual Molecular Dynamics
M	Methionine
W	Tryptophan
F	Phenylalanine
O	Hydroxyproline

CHAPTER I

INTRODUCTION

Collagen fibrils provide mechanical stability, play a large role in connective tissues, and protect the body from external physical stresses. Many defects in collagen or collagen metabolism have been connected to diseases and genetic mutations such as osteoarthritis, osteoporosis, and fibrotic diseases [1, 2]. While collagen is the most abundant protein in the body, many of its functions and processes are still not fully known and understood [3]. Biochemistry techniques such as Western Blotting and ELISA can only describe the nature of the protein before the technique begins and after it ends. The missed protein interactions that can occur or change during these techniques may be significant to better understand how proteins will interact. Since changes in protein structure and interactions may be too evanescent to process, super computers and software such as CHARMM and VMD have been used to build and analyze collagen and collagen binding proteins (CBPs). This method allows for better control and visual representations of the proteins of interest and is more accessible to people around the world. Such as with the new coronavirus [SARS-CoV-2], researchers have modeled the virus and its protein parts that allow its penetration into human cells [4]. Together with biochemistry techniques, both methods can help improve the efficiency of knowledge gained from protein interactions and diseases. Therefore, the motivation to study the structural analysis of collagen and collagen-binding proteins (CBPs) is to learn the transient structural binding processes. By first understanding the structural changes, we can further determine the physiological functions of the collagen and CBPs. This information can be used to design drugs to take advantage of the protein mechanisms in order to treat the pathology. In this article, α-1 integrin, human osteoclast- associated receptor (OSCAR), and Discoidin Domain

Receptor Tyrosine Kinase 2 (DDR2) were molecularly simulated with collagen and analyzed with software for their functions. Specifically, the α-1 domain contains the major binding site for ECM ligands. It assumes a Rossmann fold, and its ligand binding to collagen is controlled by metal ion-dependent adhesion sites (MIDASs). Together, the α-1 integrin proteins are cell surface receptors that mediate interactions between individual cells as well as bidirectional signals between cell membranes [5]. Similarly, DDR2's are widely expressed receptor tyrosine kinases that are activated by triple-helical collagen. They contain a unique discoidin domain and control important aspects of cell behavior and prevention of human diseases such as fibrosis and cancers [6]. OSCAR's function as a receptors that help regulate osteoclastic activities, which in turn are responsible for bone homeostasis. They are part of the leukocyte receptor complex family of proteins. The receptor propagates its signal by a collagen-activated signaling pathway and can activate osteoclasts, endothelial cells, and myeloid cells [7]. For more information on OSCAR, DDR2, and α-1 integrin proteins, their PDB IDs are 5CJB, 2WUH, and 2M32, respectively.

CHAPTER II

METHODS

The purpose of the study was to further understand the bound structures of collagen binding proteins. The PDB files of DDR2 [6], OSCAR [7], and α-1 integrin [5] were selected from the RCSB Protein Database [1]. These files depicted the whole proteins or the main collagen binding domains. The collagen strands used for visualization consisted of short segments of collagen-like residue sequences consisting of multiple sequences of Glycine and two additional proteins, usually proline and hydroxyproline (Gly-X-Y or Gly-Pro-Hyp). For each protein structure, a biological technique was used to determine the most accurate representation of the crystalline form of the protein. Specifically, X-ray diffraction determined a resolution of 1.6 Å for DDR2 and a resolution of 2.398 Å for OSCAR. Solution NMR and HADDOCK Docking was used to determine the structure for α-1 integrin. The PDB files contained these positional coordinate data of each atom of the protein residues as well as other information such as which residues had disulfide bonding or the presence of salt bridges.

The first step in studying the protein structures was to model the original protein coordinates. CHARMM, a force field-based simulation software, was used to create the initial protein structures, the binding domains, and the three strands of collagen. A CHARMM script was then used to merge all of the protein segments into a single structure. This was the bound configuration. The entire protein for human OSCAR and α-1 integrin were modelled. Only the binding domain of DDR2 was modelled due to its large size. CHARMM scripts were also utilized to perform solvation, neutralization, heating, and energy minimization.

In special circumstances, missing residue information among the various amino acid chains posed a problem in the calculation of a protein's structure which prevented further analysis. CHARMM normally fills in the gaps with linear connections but in the case of a loop this results in an unrealistic offshoot. MODELLER [8] was used to fill in these missing residues and statistically determine the best positions for multiple configurations. If residues were missing at the C and N terminals of the CBPs, and these residues did not affect collagen binding, then the residues were ignored.

The coordinate data for all collagen binding proteins were obtained from crystallography or NMR images. As such, all coordinates in the PDB files were estimated protein coordinates. To simulate a protein in a test tube, water was added to the simulation as a water box, such that there is at least a 20 Å gap between the protein and the water cube boundary. The water cube length for 2M32, 5CJB, and 2WUH are 90.676487 Å, 93.9827637 Å, and 99.3192884 Å respectively. The number of atoms in the protein models of 2M32, 5CJB, and 2WUH were 3830, 3696, 3574 atoms respectively. By adding TIP3 water to the models, the models more accurately depicted proteins in solution, where water can interact with the protein for higher degrees of conformational freedom and flexibility found in solution proteins.

CHARMM was also used to neutralize the charge of the proteins. 50 mM NaCl was added to the simulated aqueous environment. The sodium and chlorine ion amounts were further adjusted by altering either the sodium or chlorine ion concentrations, depending on the charge.

The next step, after building each protein in solution, was energy minimization and heating of the solution by means of CHARMM simulation. The system was subjected to a four-stage energy minimization process. At each stage, it underwent 100 steps maximum minimization followed by 300 steps of the Newton–Raphson minimization. During energy minimization, a

gradually decreasing harmonic constraint was applied to the backbone heavy atoms in the protein. Spring constants of the harmonic constraints were 5 kcal/mol·Å^2 in stage 1, 1 kcal/mol·Å^2 in stage 2, 0.1 kcal/mol·Å^2 in stage 3, and 0 kcal/mol·Å^2 in stage 4. For heating, the system's temperature was raised gradually at a rate of 2 degrees K per 100 steps up to 300 K. The heating phase took 50000 steps. Each step was .002 picoseconds. The dynamic simulation then entered into thermal equilibrium. The temperature fluctuated around 300 K over the course of 100,000 steps to allow further conformational equilibration [9].

The final step in modelling the proteins was the dynamics run. This took place at 5000000 time steps over the course of 10 nanoseconds. Again, the temperature was held at around 300 K. All dynamics data was written into coordinate and trajectory files. These files could be viewed and run using VMD for further analysis.

Further analysis included comparing the root mean square fluctuation (RMSF) of our dynamic model to the B-factor value of the original static model. The α -1 integrin, discoidin domain receptor 2 (DDR2), and human osteoclast associated receptor (OSCAR) protein were analyzed according to their root mean square fluctuation (RMSF) and square root B-factor. The RMSF values were calculated by CHARMM scripts to measure the displacement of each C-alpha atom of the protein structures, without collagen fragments, with respect to their dynamic structures as an average of all the frames. Similarly, the square root B-factor reflects the fluctuation of the atoms about their average positions in the static structures. These B-factor values were taken from the original PDB files.

We also compared the hydrogen bond occupancy of the dynamic models to the hydrogen bonds found in the static model. Occupancy levels determined the strength of the hydrogen bonding between an atom of the protein and another atom of the collagen. Hydrogen bonds in the

static model were based off of those found in the literature for the original coordinates. These bonds were checked manually by measuring the distance between the hydrogen and its nitrogen or oxygen binding partner. We used a cutoff distance of 2.4 Å to determine the presence of the hydrogen bond.

The last analysis we performed was to study the conformation of the collagen fragments. We did this by looking at the distance between the different collagen strands as well as the average angle between triads. Triads were designated based on adjacent backbone α-C atoms from each collagen strand. To eliminate the end effects, we ignored 3 residues at the C and N terminals. Each triad was a triangle, whose centroid was the origin of the triad. A normal vector was assigned at the origin and pointed at the C-terminal. Another vector was assigned at the origin that pointed between the Cα carbons of the middle and leading strand. Torsional angles could be determined by comparing the cross products of these two vectors between different triads [10]. By looking at the distance between strands and the torsional angles, we observed possible collagen unwinding behavior and attempted to determine the source of the unwinding. For further analysis of the collagen fragments' stability, we subdivided the collagen into 1 nanosecond timeframes to compare the torsional angle fluctuation over time.

CHAPTER III

RESULTS

α-1 Integrin Protein (2M32)

The α-1 integrin protein's crystal structure is shown to possess large structural changes after construction and simulation (Figure 1 (left)). After dynamic simulation, the protein was tilted along its central point, and the main interactions and hydrogen bonds seemed to lose biological accuracies by 8.3555 nanoseconds. Shown below, the collagen and α-1 integrin have become noticeably detached by 9.995 nanoseconds (Figure 1 (right)).

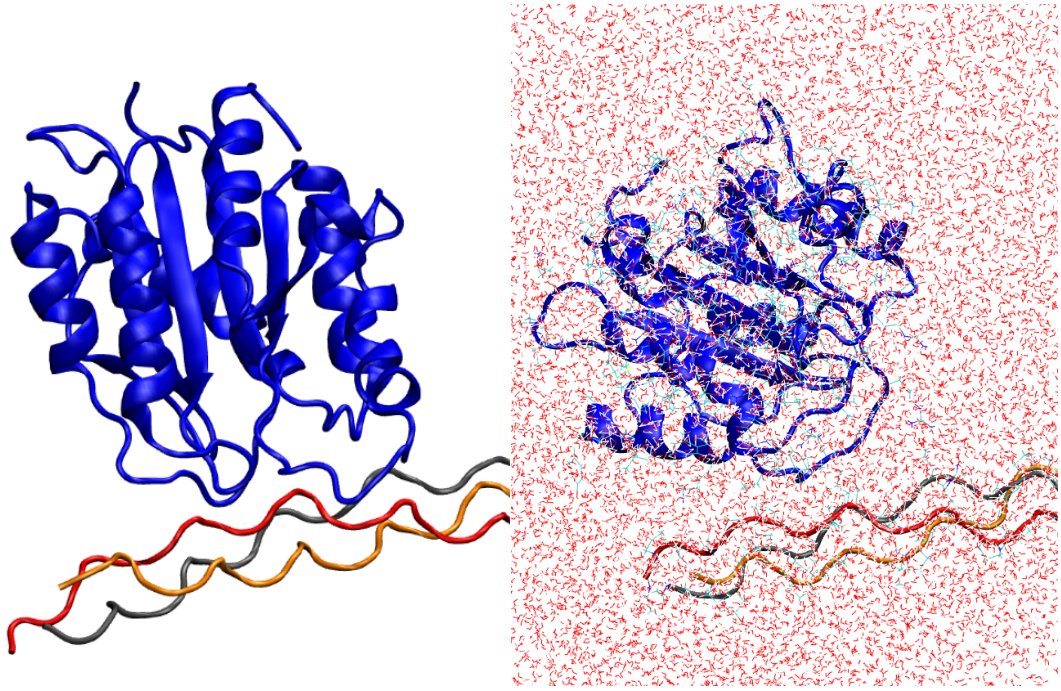


Figure 1. (left) Original structure (right) simulation structure

Furthermore, there exist many peaks of high RMSF and square root B-factor values among the red and blue lines. The dashed vertical line demonstrates that the high occupancy, the major conformation of the atom in its location, exists with a low RMSF (Figure 2). The trends indicate

that the simulation results deviate from the original static structure too much and that the results may not be realistic or provide any biological insights. In this case, the results may show that HADDOCK Docking software may not be completely accurate to help predict protein interfaces and structures. As one of the newer software used for modeling of biomolecular complexes, it may require more testing and improvement for better accurate simulations in the future.

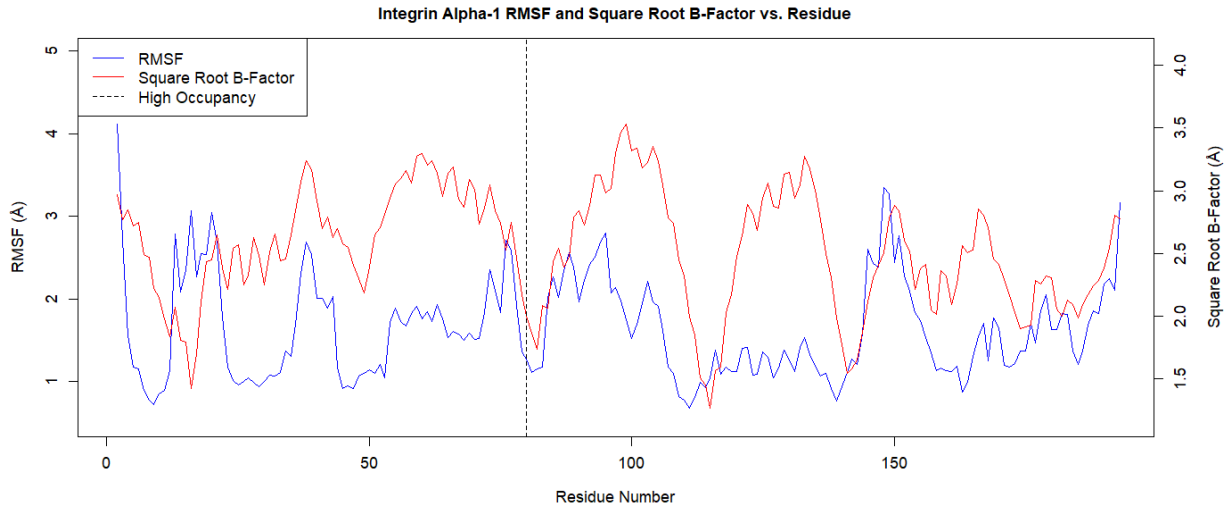


Figure 2. α-1 integrin protein's (2M32) RMSF, square root b-factor, and residue numbers

Human Osteoclast-associated Immunoglobulin-like Receptor (5CJB)

The render of 5CJB after construction and simulation in CHARMM is shown below (Figure 3 (left)). In addition, there is noticeable conformational change after energy minimization (Figure 3 (right)) when compared to the original structure in that after simulation the width of the protein was smaller and the beta sheets were angled more towards the collagen. Figure 3 (right) was taken at 6 ns.

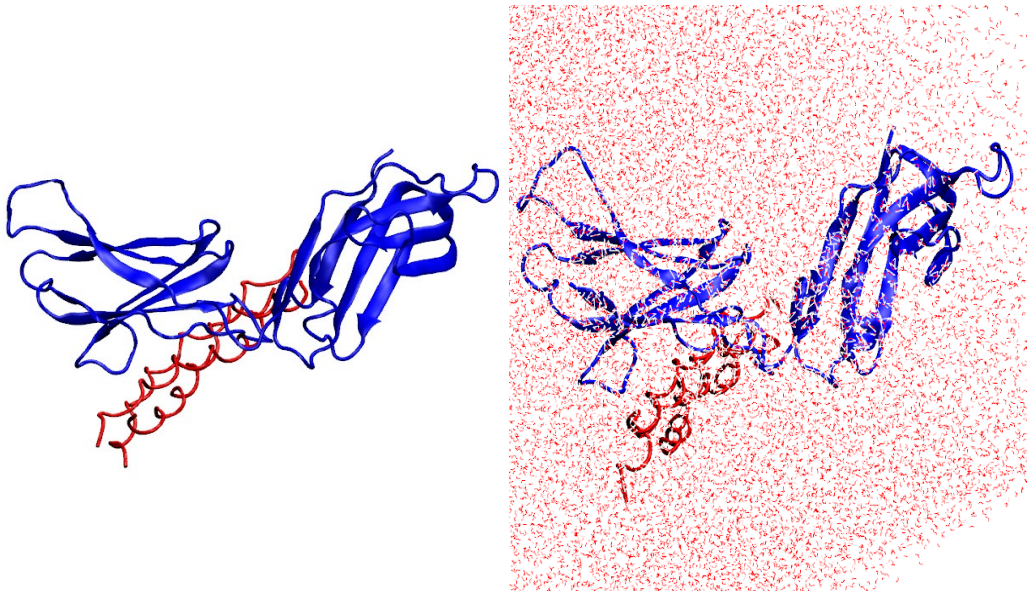


Figure 3. (left) Original structure (right) simulation structure

Comparing the RMSF and square root B-factor, there is a noticeable peak where the X-ray imaging was unable to identify the residues. This was most likely due to high variance which made identifying the residues difficult and are confirmed by the relatively high RMSF values. The lowest RMSF values coincide with high occupancy bonds as they have the least variance. The trends found in the RMSF and square root B-factor values are similar indicating the simulation has use for further analysis (Figure 4).

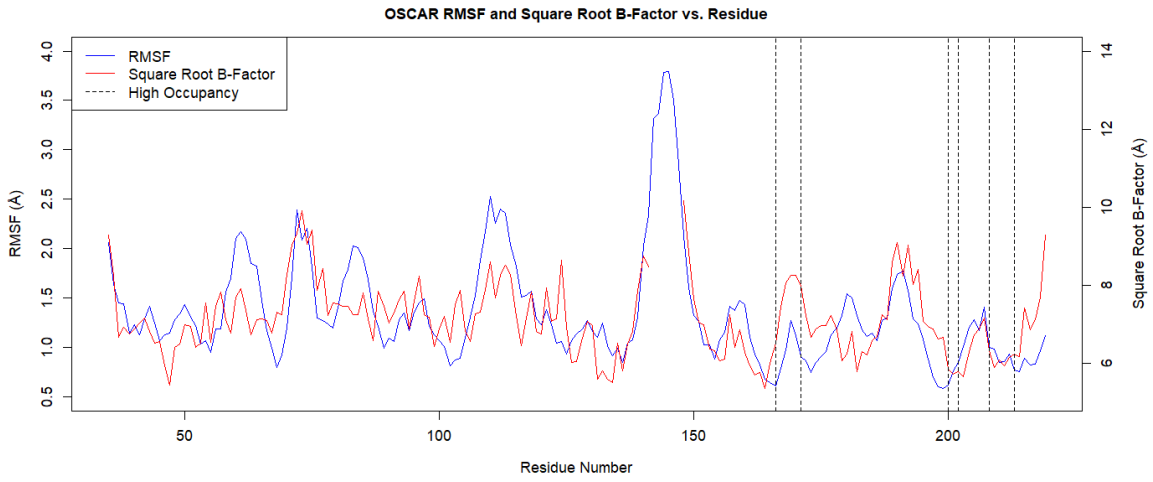
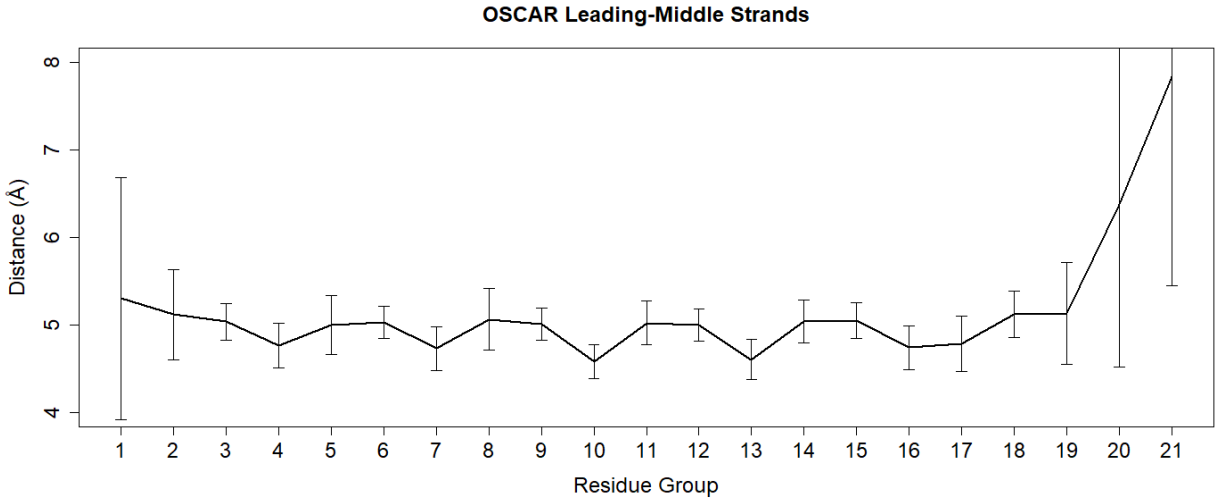
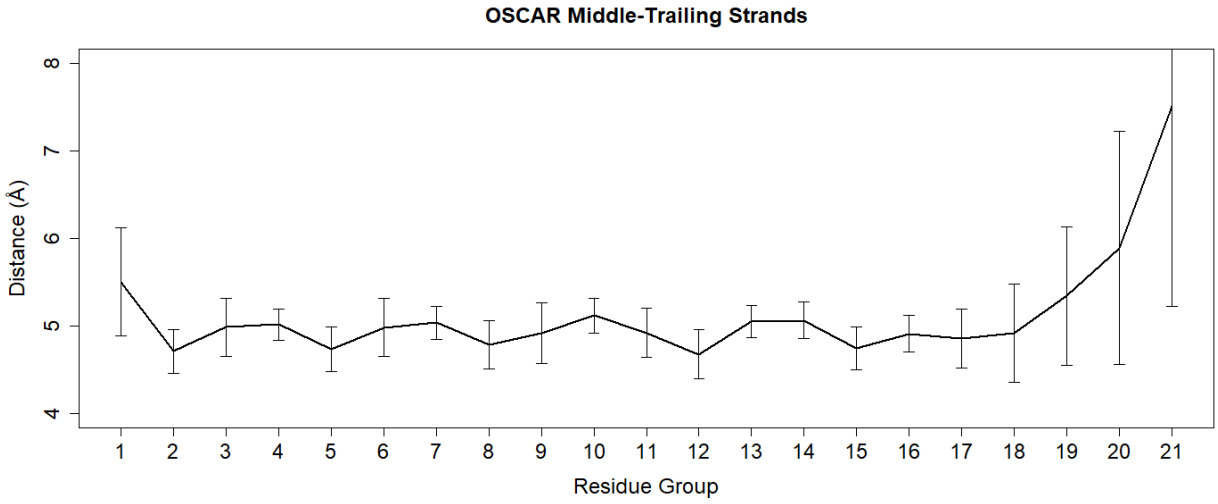


Figure 4. OSCAR's RMSF, square root b-factor, and residue numbers

Further analysis was done by determining the average distance between the alpha carbons of each of the residues in each triad. As shown below (Figure 5), the distance between alpha carbons was larger for the first and last several triads. This was most likely due to the ends of the collagen strands not being held in place during the simulation. For the central portion of the collagen structure, the average hydrogen bond distance did not waver far from about 5 Å. In the OSCAR trailing-leading strands, residue 13 has a relatively larger bonding distance from its neighboring residues. This could show that this could be the active or bonding site for the human OSCAR protein to bind.





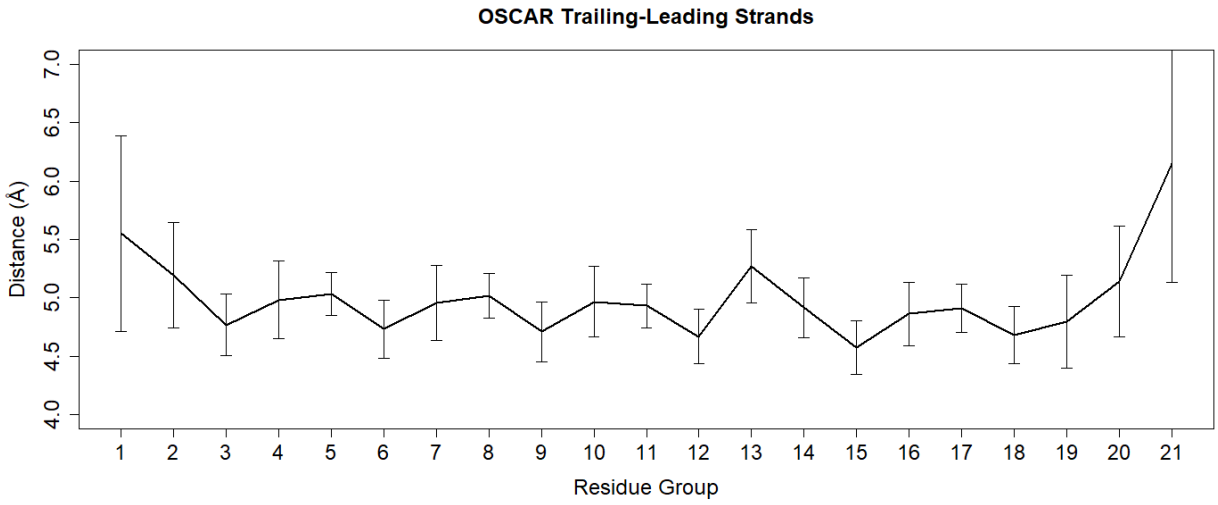


Figure 5. (top-bottom) OSCAR collagen leading, middle, and trailing strands interactions

The average torsional angle for each triad further confirms the findings in the collagen distance graphs. (Figure 6) Specifically, at triad 13, which has the lowest average torsion angle, there could be binding of Human OSCAR to the collagen chain. During the simulation, the protein did not bind with the leading strand at any point. In addition, in triads 13 and 14 the normal Gly-Pro-Hyp pattern of collagen was altered with an Ala substitution for the Hyp. Also, in triad 15 and 16, Phe was substituted for Pro. The data can be better interpreted in Figure 7, which shows the average triad torsion angle in 1 ns timeframe.

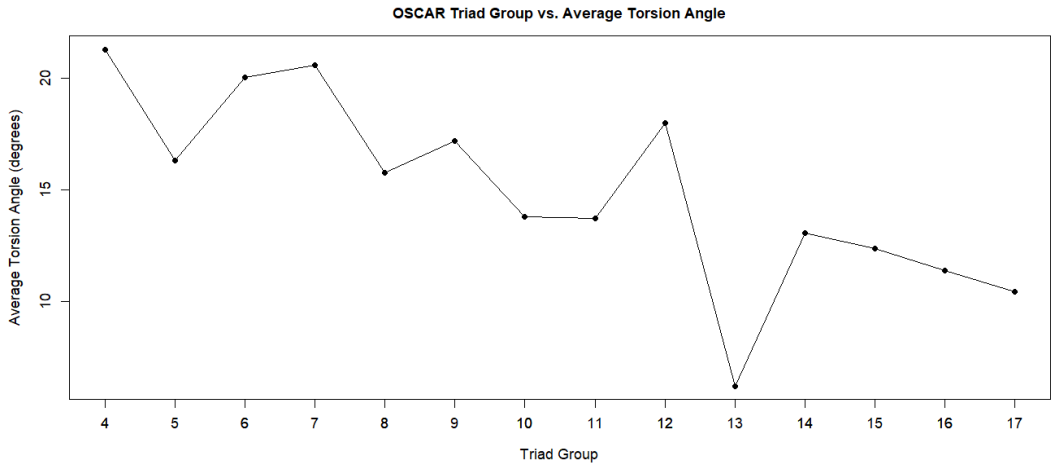


Figure 6. OSCAR correlation between triad groups and average torsional angle

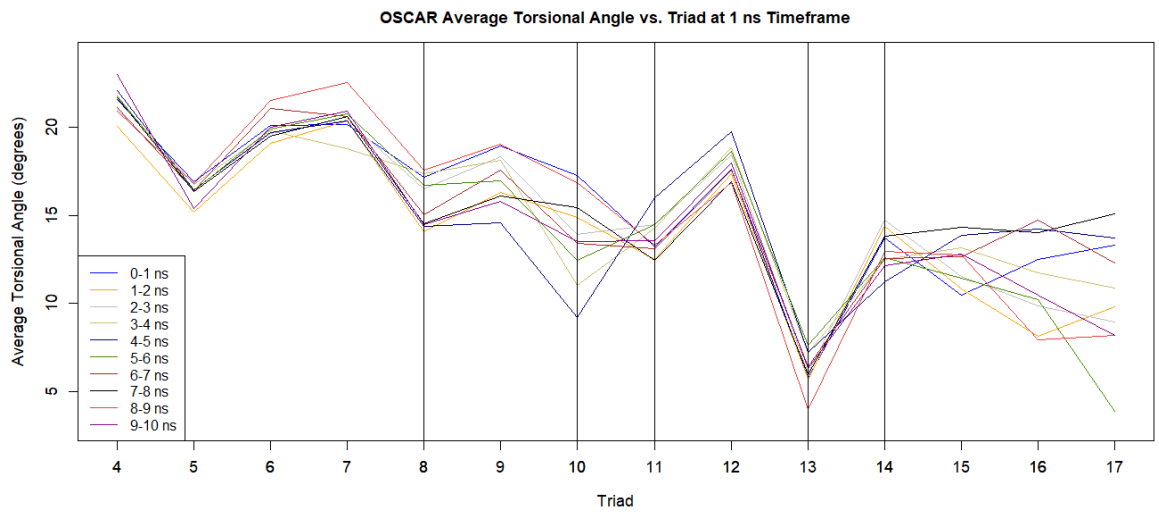


Figure 7. OSCAR average torsional angle vs. triads over time

The graph shows instability in the range of triad 8 to 10, which is most likely due to hydrogen bonding to the hydroxyproline residues in the middle and trailing collagen strands. Normally the hydroxyproline residues add stability to the collagen strands but because of the high occupancy hydrogen bonding the stability is diminished. In triad 11, the stability returns to the collagen as in this triad it contains 2 high occupancy hydrogen bonds. In triad 13, the very low average torsional angle shows significant collagen unwinding. Triad 14 is similar to triad 13 in that they both have alanine residues which bind to Human OSCAR with high occupancy hydrogen bonds but triad 14 has much less unwinding. One possible explanation for the stability despite unwinding is that the residues have multiple hydrogen bonds such as in triad 13 where an alanine residue of the middle collagen strand binds to an alanine residue of OSCAR and in triad 14 where an alanine residue of the trailing collagen strand binds to an arginine residue of OSCAR. Another possible explanation is that F17 of the trailing strand binds in a shallow pocket in Human OSCAR. This residue is present in triad 15 which may explain the increasing instability in the collagen. Overall, the variance in the average torsional angle is quite high which is most likely due to not

binding the ends of the collagen chains in place, resulting in higher movement than what would normally be expected. The high occupancy bonds are shown below at 6 ns. (Figure 8)

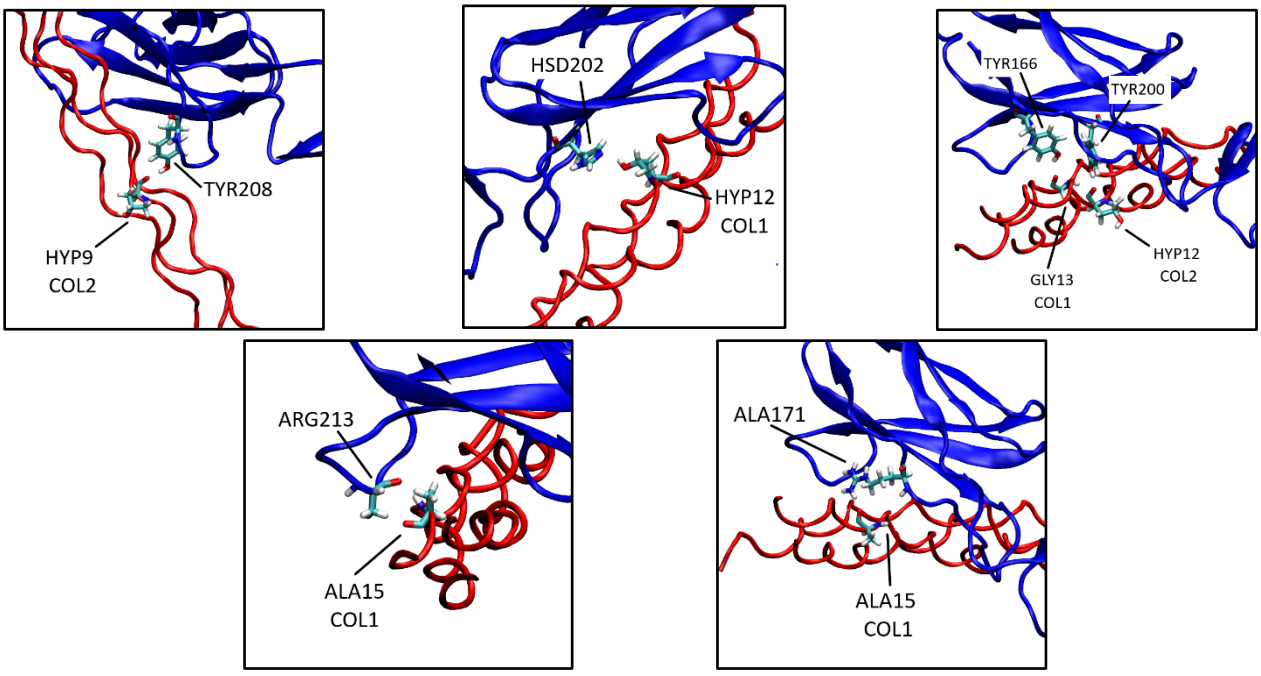


Figure 8. (top left) OSCAR triad 8, (top middle) 10, (top right) 11, (bottom left) 13, and (bottom right) 14 high occupancy bonding

Discoidin Domain Receptor Tyrosine Kinase 2 (2WUH)

The dynamic model of 2WUH (Figure 9 (right)) did not show drastic changes in structure when compared to the static model. (Figure 9 (left)) The images were taken at 9 ns into the simulation. The overall patterning between the RMSF and B-factor graph are very similar, with the larger peaks and troughs appearing in both graphs. (Figure 10) The three high occupancy hydrogen bonds were between DDR2 D69 and collagen leading strand M21, DDR2 W52 and collagen middle strand M21, and DDR2 E113 and collagen leading strand O24. These bonds were found to be present in both the static and dynamic structure. These bonds coincide with the troughs of the RMSF graph, but there some of the bonds coincide with B-factor peaks. This could be due to the estimation method used to calculate the B-value from crystallographic imaging.

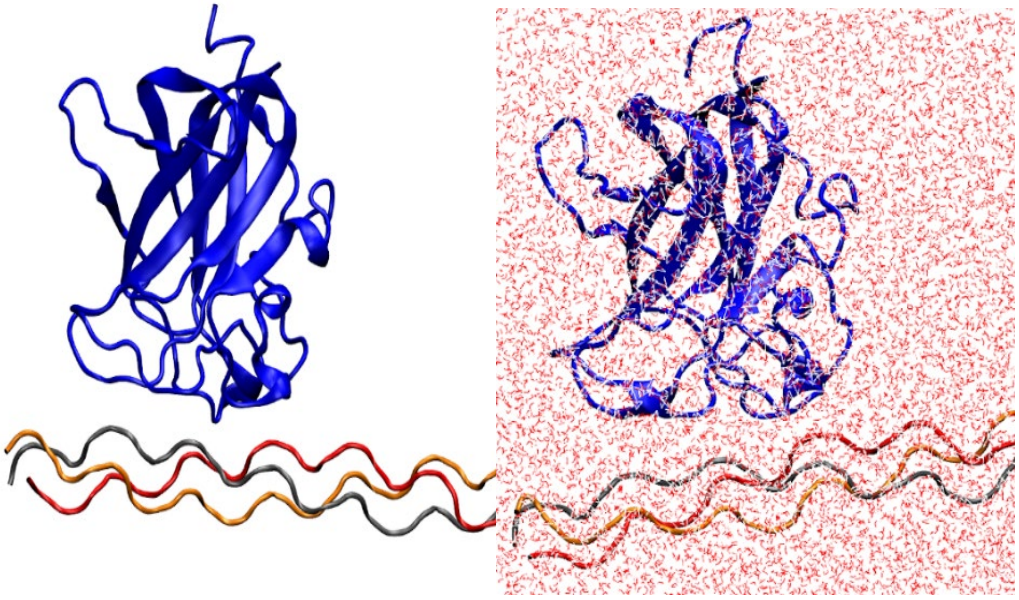


Figure 9. (left) Original structure (right) simulation structure

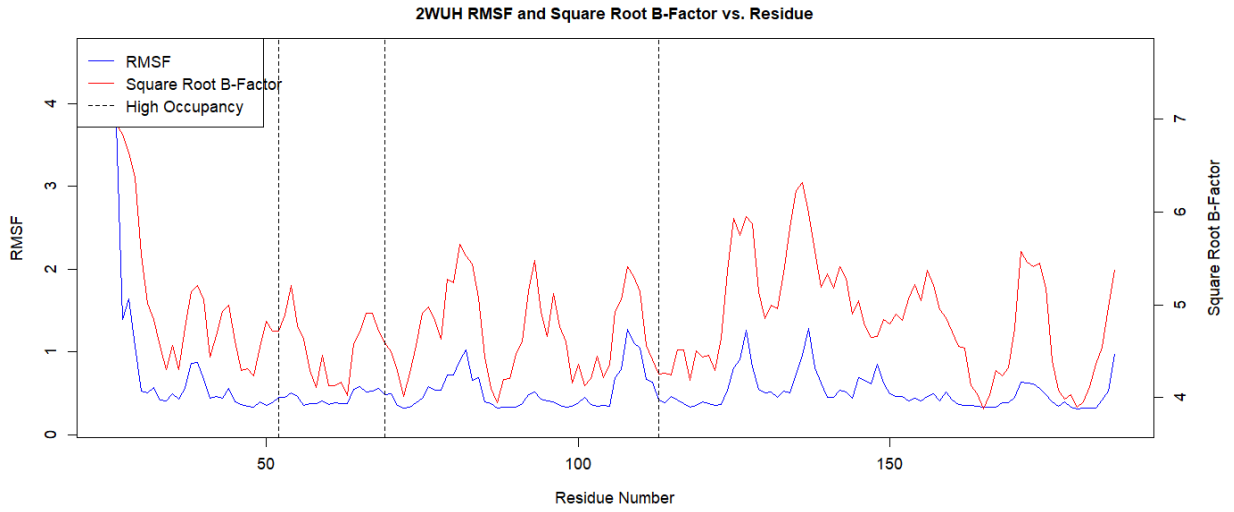


Figure 10. Discoidin domain receptor tyrosine kinase 2 (DDR2) RMSF, square root b-factor, and residue numbers

Similarly, the bonding distance was determined between each collagen strand pair for 2WUH, the DDR2 protein. The graphs below demonstrate the average distances between the different strands by residue number (Figure 11).

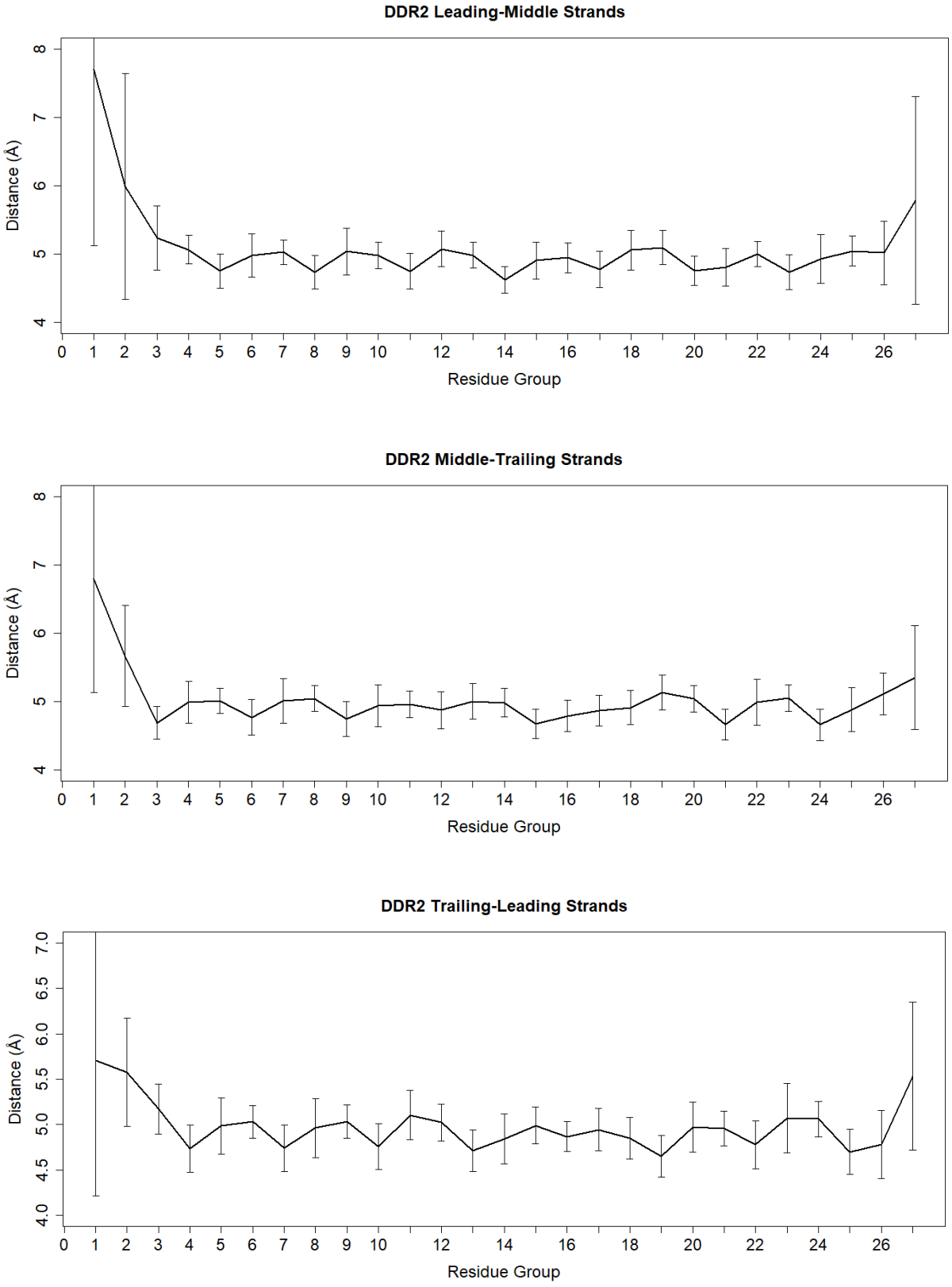


Figure 11. (top-bottom) DDR2 collagen leading, middle, and trailing strands interactions

Most of the residues, in all three graphs, do not waver significantly from an average bonding distance of 5 Å. From this, we can see that the collagen was fairly stable and there was no major dissociation between collagen strands. There was some dissociation at the ends due to the fragmentary ends of the collagen used in simulation. This may have been avoided had we simulated static end residues, such that their position was constant. However, due to the location of the binding site on collagen, we do not believe that the ends significantly changed the overall results.

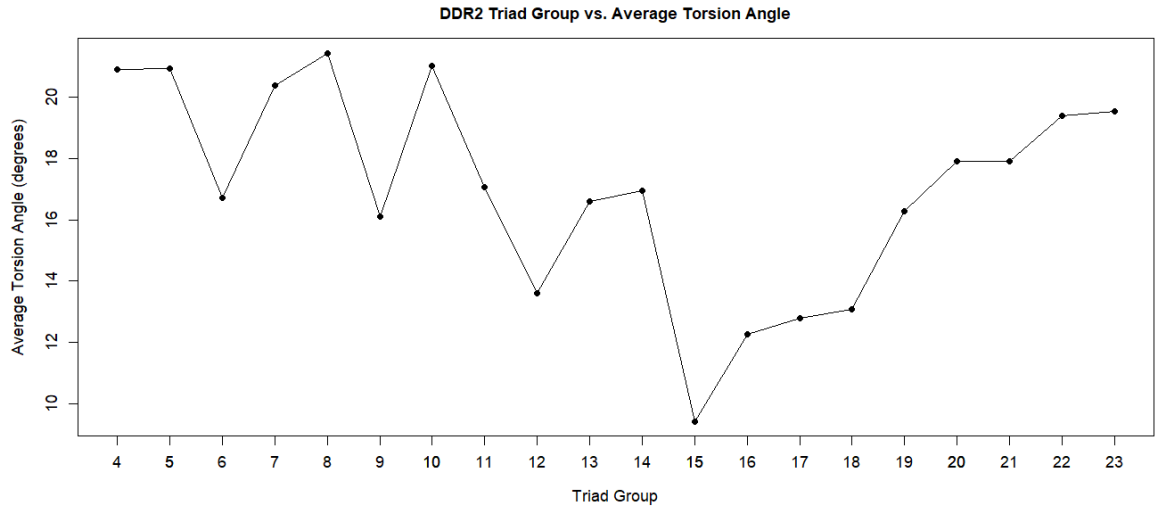


Figure 12. DDR2 correlation between triad groups and average torsional angle

What we see in the graph is a noticeable decrease in the average torsional angle starting at triad 15. (Figure 12) Triad 15 had MET21 in the leading strand that hydrogen bonded to DDR2. Triad 16 had MET21 in the middle strand that hydrogen bonded to DDR2. Triad 18 had HYP24 in the leading strand that hydrogen bonded to DDR2. The hydrogen bonding can increase the stability of the DDR2-collagen interface and increase unwinding behavior of the collagen. No lagging strand residues were hydrogen bonded to DDR2. Triad 15 is one of the three triads that bind to DDR2, but its torsional angle is lower than the other two. This may be because of the Phenylalanine residue in the middle strand. Phenylalanine is a bulky residue and may sterically

inhibit collagen winding. For further analysis of the stability of the collagen, we subdivided the frames into 1 nanosecond fragments and graphed them together.

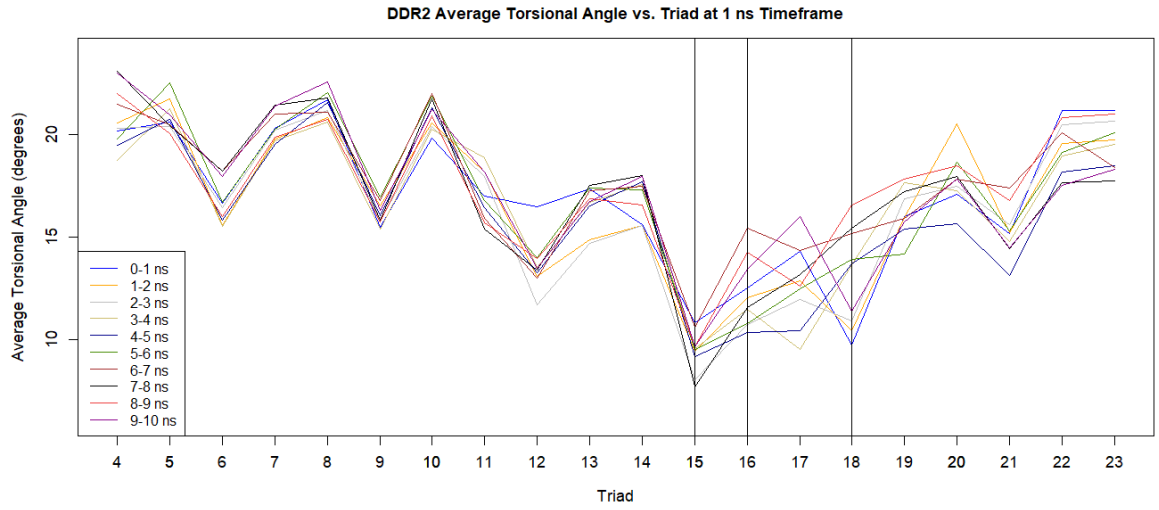


Figure 13. DDR2 average torsional angle compared to triads in 1 ns timeframe

As can be seen, up to triad 15, all time segments seem to align. (Figure 13) There is some variance at triads 12, 13, and 14, but this only occurs in early time segments. This indicates that collagen was migrating into a lower energy conformation. Triad 15 is fairly stable over time as compared to triads 16 and 18. The stability of triad 15 may be due to the bulky phenylalanine, and there is some evidence of this stability as shown above. The variance of the torsional angle with time appears to be stable. Phenylalanines in the binding region of collagen are also linked to an apparent salt bridge as described by Chin [5]. This salt bridge from the original coordinate data was also observed in the dynamic simulation and may play a further role in the unwound collagen stabilization at triad 15. (Figure 14) Triads 16's and 18's torsional angles vary greatly over time, in comparison, due to the unwound nature of the collagen segment. There was no stabilizing factor to be found besides the hydrogen bonding to DDR2. Towards the end of the collagen, triads 19 and higher, we observe some possible restabilization of triad torsional angles. What's more is that we see later time segments appear higher than earlier time segments from triads 15 to 19. After

triad 19, we start seeing the reverse, where earlier segments are higher than later segments. This appears to be due to some form of winding/unwinding motion. We believe that there are two possibilities: winding/unwinding propagation along collagen or natural reconfiguration of collagen into a lower energy conformation. These are only speculations because we do not have enough data to draw a stronger conclusion, as we only simulated the first 10 nanoseconds of dynamics.

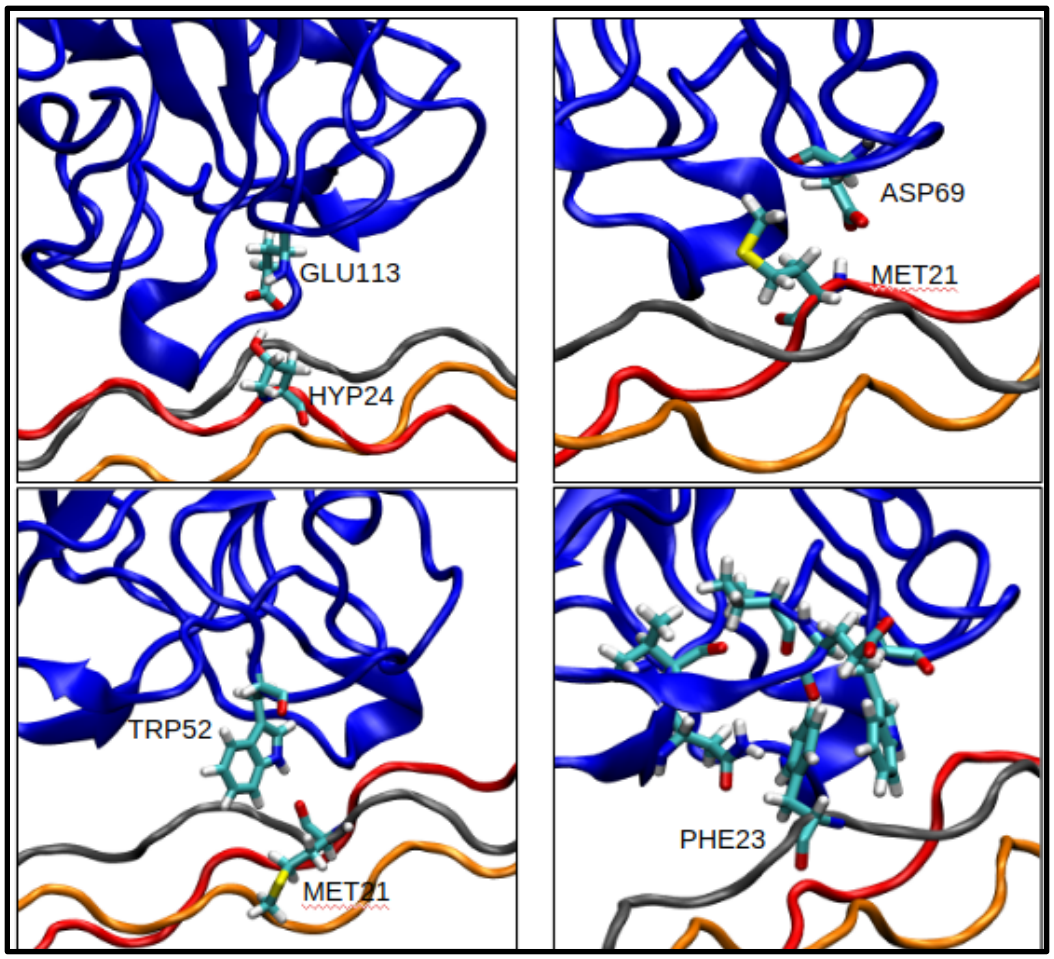


Figure 14. (top left) Strong hydrogen bonding among leading methionine, (top right) leading hydroxyproline, (bottom left) middle methionine, and (bottom right) middle phenylalanine trapped in a pocket

CHAPTER IV

CONCLUSION

After performing molecular simulation in a physiological environment, the α-1 integrin model was determined to be biologically inaccurate. The collagen and α-1 integrin were observed to break away over time, losing the interactions and functions that were intended to study. This failed binding could be due to the HADDOCK program used in the simulated binding of the original static model. As such, we halted further analysis and focused on DDR2 and OSCAR.

On the other hand, OSCAR and DDR2 dynamic models were stable during simulations. The dynamic models were similar to the static models as determined by the similarity between RMSF and square root of B-Factor graphs. (Figure 4, 10) Looking at the distance between collagen strands, we saw that there was no dissociation at the central regions. (Figures 5, 11) This means that the collagen fragments are stable, and individual strands did not dissociate.

By graphing the average torsional angles of the collagen triads, we see decreases in the torsional angle in the binding regions of both DDR2 and OSCAR collagens. This included DDR2's triad 15 and OSCAR's triad 13, whose torsional angles were significantly lower than the other triads. The change in torsional angle overtime was also graphed, and the stability of the triad angles could be observed. (Figures 7, 13) There was a significantly increased stability in 2WUH's triad 15 and 5CJB's triads 11 and 14 as shown by the reduced variability. We believe this unwinding behavior and changes in stability are due to the triads' interactions with DDR2 and OSCAR.

From this data, we can determine that both DDR2 and OSCAR simulations have stable binding to their respective collagen fragments. These interactions seem to be related to the unwinding of collagen. Generally, these unwound portions of collagen were unstable over time.

However, there seemed to be increased stability in these unwound regions when triads of collagen had multiple hydrogen bonds to the DDR2/OSCAR or some kind of steric stabilization in a protein pocket. DDR2 and OSCAR unwind collagen when binding, and certain interactions between the proteins and collagen can stabilize the unwinding behavior.

REFERENCES

[1] H.M. Berman, J. Westbrook, Z. Feng, G. Gilliland, T.N. Bhat, H. Weissig, I.N. Shindyalov, P.E. Bourne. The Protein Data Bank Nucleic Acids Research 28, 235-242 (2000).

[2] Shoulders, Matthew D., and Ronald T. Raines. "Collagen Structure and Stability." Annual review of biochemistry 78, 929-58 (2009).

[3] Theocharis, A., Skandalis, S. Gialeli, C., & Karamanos, N. Extracellular Matrix Structure. Advanced Drug Delivery Reviews 97, 4-27 (2016).

[4] Wrapp, Daniel, et al. "Cryo-Em Structure of the 2019-Ncov Spike in the Prefusion Conformation." Science 367.6483 (2020): 1260. Print.

[5] Chin, Y. K.-Y., et al. The Structure of Integrin α1I Domain in Complex with a Collagen-mimetic Peptide. Journal of Biological Chemistry 288, 36796–36809 (2013).

[6] Carafoli, F., et al. Crystallographic Insight into Collagen Recognition by Discoidin Domain Receptor 2. Structure 17, 1573-1581 (2009).

[7] Haywood, J., et al. Structural basis of collagen recognition by human osteoclast-associated receptor and design of osteoclastogenesis inhibitors. Proceedings of the National Academy of Sciences 113, 1038–1043 (2016).

[8] Eswar, N., et al. Comparative Protein Structure Modeling Using Modeller. Current Protocol Bioinformatics 0 5, Units 5-6 (2006).

[9] Cho, J., et al. Molecular recognition of a host protein by NS1 of pandemic and seasonal influenza A viruses. PNAS 117, 6550-6558 (2020).

[10] Teng, X. and Hwang, W. Chain Registry and Load-Dependent Conformational Dynamics of Collagen. Biomacromolecules 15, 3019-3029 (2014).

[11] Wonmuk, H., Jang, M., & Karplus, M. Kinesin motility is driven by subdomain dynamics. eLife, 1-25 (2017).

[12] Teng, X. and Hwang, W. Effect of Methylation on Local Mechanics and Hydration Structure of DNA. Biophysical Journal 114(8), 1791-1803 (2018).

APPENDIX I

MERGE

```
* merge.inp
*

! script will combine protein and collagen into one file

! set error level
bomlev -1

! open general parameters
stream include/include.str

! open protein psf and cor files
read psf card name oscar.psf
read coor card name oscar.cor

! open collagen psf and cor files
read psf append card name coll.psf
read coor append card name coll.cor

! combine protein and collagen
write psf card name 5cjb.psf
* 5cjb
*
write coor card name 5cjb.cor
* 5cjb
*

stop

```

APPENDIX II

SOLVATE

```
* solvate1.inp: puts water in a cubic box with edge length the larges
* required.
* This code can be used for any molecule. It runs 3x faster than solvate.inp
* Below made y and z size of the box the same, whle the long axis of the
* molecular lies along the x-dir (longest)
* Usage: charmmxxl < solvate1.inp > out_solvate1.dat
*

! set error level
bomlev -1

! include basic parameters
stream ~/include/include.str

! set input and output filenames
set I 5cjb ! input filename
set O 5cjb_sol   ! output filename

ioform extended

! Asymmetric cutoffs to allow motion of cs & nl if detach
set cutx 13.0 ! cutoff for making water box
set cuty 13.0 ! cutoff for making water box
set cutz 13.0 ! cutoff for making water box

! information of the unit water box
set L  37.712 ! size of unit water boxs, unit of length: Angstrom
calc L1 0.5*@L ! half the size of unit water box
set nwat 1728 ! 216 ! number of water in unit box

! Read the sequence from PSF file
read psf card name @I.psf

! Read the coord file
read coor card name @I.cor

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! Use coor rota below to orient the molecule adequately in water box
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

coor orie mass
coor stat mass

! In the following, coor rota should yield similar [x,y,z][min,max] in
! the output of coor stat

!coor rota xdir 1.0 ydir 0. zdir 0. phi -45 sele all end
coor rota xdir 0.0 ydir 1. zdir 0. phi 41 sele all end
coor rota xdir 0.0 ydir 0. zdir 1. phi 60 sele all end
coor stat

!write coor pdb name temp_orie.pdb
!* @I_m0 after coor orie (can be deleted; for determining orientation of
!* the molec in the box
!*

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

! box size
calc dx ?xmax - ?xmin + 2* @{cutx}
calc dy ?ymax - ?ymin + 2* @{cuty}
calc dz ?zmax - ?zmin + 2* @{cutz}
```

```
63
64  set lbox @{dx}
65  if @{lbox} .lt. @{dy} set lbox @{dy}
66  if @{lbox} .lt. @{dz} set lbox @{dz}
67
68  set dx @{lbox}
69  set dy @{lbox}
70  set dz @{lbox}
71
72  !Make dy & dx the same (take the larger)
73  !F @{dy} .gt. @{dx} set dz @{dy}
74  !F @{dy} .lt. @{dx} set dy @{dx}
75
76  ! half the box size
77  calc dx1 0.5*@{dx}
78  calc dy1 0.5*@{dy}
79  calc dz1 0.5*@{dz}
80
81  ! number of unit boxes in each direction
82  CALC nx INT ( @{dx} / @L ) +1
83  CALC ny INT ( @{dy} / @L ) +1
84  CALC nz INT ( @{dz} / @L ) +1
85
86  !stop ! first pass
87
88  !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
89  !  Build a big cube to surround solute
90  !prnlev 0 node 0
91
92  ! corner of the water box
93  calc xpos0 @{L1} - @{dx1}
94  calc ypos0 @{L1} - @{dy1}
95  calc zpos0 @{L1} - @{dz1}
96
97  ! Read unit water box
98  read sequ tip3 @{nwat}
99  generate w0 setup noangle nodihe
100
101 ! for tip216.crd
102 !open unit 1 read form name @{d0}tip@{nwat}.crd
103
104 ! for wat1728.cor
105 open unit 1 read form name "~/include/wat1728.cor"
106 read coor card unit 1 append
107 close unit 1
108
109 ! Translate the original box
110 coor tran xdir @{xpos0} ydir @{ypos0} zdir @{zpos0} sele segi w0 end
111
112 ! expansion in x-dir
113 set seg 1
114 set xpos @L
115 label xmov
116   gene w@{seg} dupli w0 setup
117   coor dupli sele segi w0 end sele segi w@{seg} end
118   coor trans xdir @{xpos} ydir 0. zdir 0. sele segi w@{seg} end
119   calc xpos @{xpos} + @L
120   incr seg by 1
121   if @{seg} .lt. @{nx} goto xmov
122 ! Delete atoms out of range and join segments
123 dele sort atom sele .byres. (segi w* .and. prop X .gt. @{dx1} ) end
124 set seg 1
```

```
label xjoin
  join w0 w@{seg} renum
  incr seg by 1
if @{seg} .lt. @{nx} goto xjoin

! expansion in y-dir
set seg 1
set ypos @L
label ymov
  gene w@{seg} dupli w0 setup
  coor dupli sele segi w0 end sele segi w@{seg} end
  coor trans xdir 0. ydir @{ypos} zdir 0. sele segi w@{seg} end
  calc ypos @{ypos} + @L
  incr seg by 1
  if @{seg} .lt. @{ny} goto ymov
! Delete atoms out of range and join segments
dele sort atom sele .byres. (segi w* .and. prop Y .gt. @{dy1} ) end
set seg 1
label yjoin
  join w0 w@{seg} renum
  incr seg by 1
if @{seg} .lt. @{ny} goto yjoin

! expansion in z-dir
set seg 1
set zpos @L
label zmov
  gene w@{seg} dupli w0 setup
  coor dupli sele segi w0 end sele segi w@{seg} end
  coor trans xdir 0. ydir 0. zdir @{zpos} sele segi w@{seg} end
  calc zpos @{zpos} + @L
  incr seg by 1
  if @{seg} .lt. @{nz} goto zmov
! Delete atoms out of range and join segments
dele sort atom sele .byres. (segi w* .and. prop Z .gt. @{dz1} ) end
set seg 1
label zjoin
  join w0 w@{seg} renum
  incr seg by 1
if @{seg} .lt. @{nz} goto zjoin

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

! Delete water atoms close to solute
define solute select .not. segid w0 end
!  Remove water overlaps with solute system
delete sort atom sele .byres. ( segid w* .and. .not. segid WAT0 .and. type OH2 .and. -
                  (( solute .and. .not. hydrogen ) -
                  .around. 2.8 )) end

prnlev 5 node 0
coor stat

write psf card name @O.psf
* solvated structure
*

write coor card name @O.cor
* solvated structure
*


```

```
187 stop
188
```

APPENDIX III

BUILD BOX

```
* pbc.str: Apply pbc
*

! define the dimensions and corners of box

! Center of the image
set xc 0.0
set yc 0.0
set zc 0.0
set L 93.9827637

crystal define cubic @L @L @L  90.0 90.0 90.0

!crystal define orth @{lx} @{ly} @{lz}  90.0 90.0 90.0

crystal build cutoff 16 noper 0

```

APPENDIX IV

NBOND

```
* nbond.str: Use when PBC is used w/ constant vol.
*

image byres xcen @{xc} ycen @{yc} zcen @{zc} sele resn TIP3 .or. resn MG -
      .or. resn SOD .or. resn POT .or. resn CLA .or. resn ZN2 .or. resn CAL   end

! PME
! kappa=5/cutofnb, fftx, etc: close to box length, but multiple
! of powers of 2, 3, 5 (ewald.doc)

nbonds atom cdiel -
    vatom vshift cutim 13.0 wmin .5 -
    cutnb 13.0 ctofnb 12. ctonnb 8.0 BYCB -
    ewald pmewald kappa 0.41 spline order 6 fftx 90 ffty 90 fftz 90

```

APPENDIX V

NEUTRALIZATION

```
* Neutralize system
*

bomlev -1

! make sure include folder is in current working directory
stream ~/include/include.str

set I 5cjb_sol ! input filename
set O 5cjb_neu   ! output filename



ioform extended

!format                          ! reset formating
set mnd 5.5                     ! minimum distance to solute, other ions
! set sol .not. ( segid w0 .or. segi tip3 ) ! atoms selection for solvent
set emin 1E20                   ! initial min energy value
set ncfg 1                      ! initialize loop counter
set last 3                      ! no. of passes thru the loop
random uniform iseed 314159     ! change iseed to sample diff states

! read the structure
read psf card name @I.psf
read coor card name @I.cor

set watseg W0  ! define solvent

! set the protein segname
set protseg  ( segi DDR2 .or. segi ZN2 .or. segi CAL .or. segi COL1 .or. segi COL2  .or. segi COL3
.or. segi WAT0  )  ! protein segment

set qtot ?CGTOT


! cations to add
! different segnames for added ions since original pdb may have them

! add Sodium Na+
set qpseg1 NA
set nqp1 50   ! number of respectiveions
set qp1 SOD   !resname



! add Chloride Cl-
set qn1 CLA                   ! set resname of neg. ion
set qnseg1 CLI
set nqn1 50

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! BEGINNING OF MAIN MONTE-CARLO LOOP
label PUTION
time now ! show current time (later can be used to get elapsed time)


! re-read the initial PSF and CRD at the beginning of the main loop
read psf card name @I.psf
read coor card name @I.cor

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

```
! RANDOM WATER REPLACEMENT
set qseg @{qpseg1}
set qatom @{qp1}
set nchg @{nqp1}
!define xcld sele .not. segi @{watseg} .or. segid @{qseg} end
stream ~/include/addions.str


set qseg @{qnseg1}
set qatom @{qn1}
set nchg @{nqn1}
!define xcld sele .not. segi @{watseg} .or. segid @{qpseg1} -
!       .or. segid @{qpseg2} .or. segid @{qnseg1} end
stream ~/include/addions.str

! RENUMBER THE WATER MOLECULES as some are deleted
join @{watseg} renum

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!! Brief energy minimization to check if ion placement is acceptable
stream pbc.str ! define  pbc (bun not non-bonded energy yet)
stream nbond.str

! fix solute:
cons fix sele @{protseg} end

! BRIEF MIN OF IONS INSERTED INTO SOLVATED MODEL
mini sd nstep 20 nprint 5

!! abnr doesn't work, probably because sd step is too small
!mini abnr step 0.05 nstep 20 nprint 5

! if the current energy is higher than the previous one, do not write the
!   structure and jump to the test for exit

if emin .lt. ?ENER goto test

cons fix sele none end

! WRITE THE LATEST MINIMUM ENERGY RESULT; current CONFIG ACCEPTED
write psf card name @O.psf
* @O (with added ions)
*
! DO AN UPDATE AND WRITE THE COOR FILE
!update

write coor card name @O.cor
* @O (with added ions)
*

! UPDATE MINIMUM ENERGY
set emin ?ENER

! TEST FOR EXIT, AND SETUP FOR NEXT PASS; REVERT TO STDOUT, CLOSE LOG
label test
crystal free
shake off
incr ncfg by 1
time diff

!outu 6
!close unit 10
```

```
124 
125 if @{ncfg} .le. @{LAST} goto PUTION
126 
127 stop
128 
```

APPENDIX VI

ENERGY MINIMIZATION

```
* energy minimization
*

bomlev -1

! read include folder in home directory
stream ~/include/include.str

! input filename
set I 5cjb_neu
! output filename
set O 5cjb_min

!! PBC.str
! Center of the image
set xc 0.0
set yc 0.0
set zc 0.0

! user defined length of box (check PBC.str)
set L 93.9827637

! length and angles
crystal define cubic @L @L @L  90.0 90.0 90.0

! read input psf and cor files
read psf card name psf/@I.psf
read coor card name cor/@I.cor

crystal build cutoff 16 noper 0

! same file from solvation
stream nbond.str

! water and ions are defined as solids
define solvent sele segi w* .or. segi MGI .or. segi NA .or. segi CLI  .and. .not.  segi WAT0 end

! standard backbone
set bb (  type CA .or. type C .or. type O .or. type N )  ! backbone

! all protein segment names
set prot ( segi OSCA .or. segi COL1 .or. segi COL2 .or. segi COL3 )

! first fix protein and minimize water & ions ==========
cons fix sele .not. solvent end
mini sd step 0.01 nstep 500 nprint 500
mini abnr step 0.05 nstep 500 nprint 500
cons fix sele none end


cons harm force 5 sele @{prot} .and. @{bb} end
mini sd step 0.01 nstep 100 nprint 100
mini abnr step 0.05 nstep 300 nprint 300
cons harm clear

cons harm force 1 sele @{prot} .and. @{bb} end
mini sd step 0.01 nstep 100 nprint 100
mini abnr step 0.05 nstep 300 nprint 300
cons harm clear

cons harm force 0.1 sele @{prot} .and. @{bb} end
mini sd step 0.01 nstep 100 nprint 100
```

```
mini abnr step 0.05 nstep 300 nprint 300
cons harm clear

! minmize everything
mini sd step 0.01 nstep 100 nprint 100
mini abnr step 0.05 nstep 300 nprint 300

prnlev 3 node 0

writ coor card name cor/@0.cor
* @0 (with added ions), initial energy minimized
*

stop

```

APPENDIX VII

HEATING AND EQUILIBRATION

```
* heat_eq.inp: heating & equilibration combined
* revised by Jie Shi, for c38a13  for ADA heat& eq 2
*

bomlev -1

set I 5cjb_neu  !input psf file name
set I1 5cjb_min  !input cor file name

set O 5cjb_   !output file name

set bb (  type CA .or. type C .or. type O .or. type N )  ! backbone
set prot ( segi OSCA .or. segi COL1 .or. segi COL2 .or. segi COL3 ) ! protein

! read forcefield files
stream ~/include/include.str

!! PBC
! Center of the image
set xc 0.0
set yc 0.0
set zc 0.0
set L 93.9827637

crystal define cubic @L @L @L  90.0 90.0 90.0

read psf card name psf/@I.psf
read coor card name cor/@{I1}.cor

crystal build cutoff 16 noper 0

stream nbond.str

! calculate pmass and tmass

calc PMASS  INT ( ?masst * 0.02  )
calc TMASS  INT ( ?masst * 0.2  )

! domdec
ENERGY DOMD

prnlev 3 node 0

shake fast bonh para

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

! restrain backbone of protein
set hf 1 ! harmonic spring const
cons harm force @{hf} sele @{prot} .and. @{bb} end


! Heating
!https://charmmtutorial.org/index.php/MD
!https://www.charmmtutorial.org/index.php/Molecular_Dynamics
OPEN WRIT UNIT 40 CARD NAME rst/@Oh.rst
OPEN WRIT UNIT 41 FILE NAME dcd/@Oh.dcd

prnlev 2 node 0

DYNA STRT cpt NSTEP 50000 TIME 0.002 IPRFRQ 50 IHTFRQ 100 -
IEQFRQ 100 NTRFRQ 50 INBFRQ -1 IHBFRQ 0 ILBFRQ 0 imgfrq -1 -
```

```
IUNREA -1 IUNWRI 40 IUNCRD 41 NSAVC 2000 NSAVV 0 -
NPRINT 5000  FIRSTT 0.0 FINALT 300 TEMINC 2 echeck -1 -
TWINDH 5.0 TWINDL -5.0 IASORS 1 IASVEL 1 ICHEW 0 -
!pconstant pmzz 225.0 pmxx 0.0 pmyy 0.0 pref 1.0
pconstant pmass @{PMASS}  pref 1.0 pgamma 20

prnlev 5 node 0

cons harm clear

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! equilbration

! restrain backbone of protein
set hf 0.5 ! harmonic spring const
cons harm force @{hf} sele @{prot} .and. @{bb} end

OPEN read UNIT 31 CARD NAME rst/@Oh.rst
OPEN WRIT UNIT 32 CARD NAME rst/@Oequ.rst
OPEN WRIT UNIT 33 FILE NAME dcd/@Oequ.dcd

prnlev 2 node 0

DYNA restart cpt NSTEP 100000 TIME 0.002 IPRFRQ 50 IHTFRQ 0 -
IEQFRQ 100 NTRFRQ 50 INBFRQ -1 IHBFRQ 0 ILBFRQ 0 imgfrq -1 -
IUNREA 31 IUNWRI 32 IUNCRD 33 NSAVC 2000 NSAVV 0 -
NPRINT 10000  FIRSTT 300.0 FINALT 300 TEMINC 0 - !echeck -1 -
TWINDH 5.0 TWINDL -5.0 IASORS 0 IASVEL 0 ICHEW 0 -
!pconstant pmzz 225.0 pmxx 0.0 pmyy 0.0 pref 1.0
pconstant pmass @{PMASS}  pref 1.0 pgamma 20

cons harm clear

prnlev 5 node 0

write coor card name cor/@Oequ.cor
* Coordinates after heat&quilibrium
*

stop

```

APPENDIX VIII

DYNAMIC/PRODUCTION RUN

```
* dyn_1.inp: production run
* by Jie: 12/10/2014
*
bomlev -1
set I 5CJB_neu   ! input psf file name
set I1 5CJB_equ ! input rst file name

set O 5CJB_1   !output file name


stream ~/include/include.str

! pbc.str
! Center of the image
set xc 0.0
set yc 0.0
set zc 0.0
set L 93.9827637

crystal define cubic @L @L @L  90.0 90.0 90.0

! read input psf, rst file
read psf card name psf/@I.psf
read coor dynr curr resi name rst/@{I1}.rst

! build the crystal structure
crystal build cutoff 16 noper 0

stream nbond.str

ENERGY DOMD

prnlev 3 node 0

shake fast bonh para

! start production run

OPEN read UNIT 31 CARD NAME rst/@{I1}.rst
OPEN WRIT UNIT 32 CARD NAME rst/@O.rst
OPEN WRIT UNIT 33 FILE NAME dcd/@O.dcd

prnlev 2 node 0

DYNA REST cpt leap NSTEP 5000000 TIME 0.002 IPRFRQ 1000 NTRFRQ 1000 -
  INBFRQ 2 IMGFRQ 2 ECHECK -1 hoover TMASS 1000.0 REFT 300.0 NPRINT 500 -
  IUNREA 31 IUNWRI 32 IUNCRD 33 NSAVC 2500 NSAVV 0 ichecw 0


write coor card name cor/@O.cor
* Coordinates after simualtion of all atom
*


stop

```

APPENDIX IX

GET BONDS

This code extrapolates the values of the bonds [11].

```
/* get_bond.cpp: find contacts from the output of internal_{hb,np}.inp.
  This replaces get_hb.py, which had memory limitation.
  Usage:
  g++ get_bond.cpp -O3 -o te
  ./te input.dat
  input.dat: input configuration file, containing:

  ifname : output of internal_{hb,np}.inp (out_internal_{hb,np}.dat)
  read_bond_data: If present read the corresponding file that is generated by
    previous execution of this program.
    If read_bond_data is present:
     - ifname is ignored.
     - Bond occupancy for the time specification (frm_ini,frm_fin,stride) is
       calculated.
  ofname: output file prefix
  btype: bond type. 1) hbond (default) 2) nonpoloar

  npad: number of frames for calculating local occupancy
  lcut/hcut: occupancy cutoffs for identifying transitions
  * npad<0, occupancy trajectory is not calculated.
  * npad<0 || lcut<0 || hcut<0:  transition is not calculated

  dt: coord saving frequency
  frm_ini: Initial frame (default: 0)
  frm_fin: final frame (default: last)
  stride: number of frames to skip (default: 1)
  * for calculating occupancy trajectory and transition, whole trajectory is
    used, and  frm_{ini,fin}, stride are ignored .

*/

#include <iostream>
#include <fstream>
#include <sstream>
#include <cassert>
#include <cstdio>
#include <cmath>
#include <iomanip>
//#include <iterator>
//#include <string>
#include <cstring>
#include <cstdlib>
//#include <list>
#include <map>
using namespace std;
#define maxbond 5000
#define maxframe 80000  //jie

void getavg(double *avg, double *sig, double ia[], int N);
void getminmax(double *min, int *imin, double *max, int *imax,
               double ia[], int N);

/*************************************************************************/
class contact{
public:
  int nframe,nbond,ntrans;
  int resi1[maxbond],resi2[maxbond];
  int npad,frm_ini,frm_fin,stride; // npad: number of initial/final frames for padding
  double lcut,hcut,hcut0,dt;
  string resn1[maxbond],resn2[maxbond],segi1[maxbond],segi2[maxbond];
  string ifname,ofname,btype; //,occ_name;
  string trajname,segname;
```

```
  char **traj;
  double **occ_traj; // ocupancy trajectory
  multimap<double,int> occ_bond; // int: bond number
  multimap<int,double> bond_occ; //
  // Transition related:
  // transition: bond number and transition frame number
  // trans_occ: bond number and occupancy before/after tran
  // trans_occ_std: std of the local avg occupancy before/after transition
  // occupancy <0: bond breakage. >0: bond formation
  multimap<int,int> transition;
  multimap<int,double> trans_occ,trans_occ_std;
  // Member function
  contact (); // constructor
  double calculate_occupancy(int i0, int n1, int n2);
  void get_args(string cfname);
  void get_occupancy();
  void get_occ_traj();
  void get_transition();
  void read_bond_data();
  void read_data();
  void read_header(ifstream& ff);
  void write_data();
  void write_header(ofstream& ff);
  void write_occupancy();
  void write_occ_traj();
  void write_transition();
};

/*************************************************************************/
contact::contact() { // constructor
  int i,j;
  //traj=new char*[maxbond];
  //  occ=new double[maxbond];
  for (i=0;i<maxbond;i++) {
    resi1[i]=resi2[i]=0; //traj[i]=new char[maxframe];
    //for (j=0;j<maxframe;j++) traj[i][j]=0;
  }
  nframe=nbond=0;
}

/*************************************************************************/
double contact::calculate_occupancy(int i0, int n1, int n2)
{ /* calculate occupancy of bond i0 in interval [n1,n2) */
  int j;
  double rdum=0.;
  for (j=n1;j<n2;j++) rdum+=(double)traj[i0][j];
  return rdum/(double)(n2-n1);
}

/*************************************************************************/
void contact::get_args(string cfname)
{
  ifstream ff(cfname.c_str());
  string sdum1; char cdum[256];
  //ifname=ofname=occ_name=btype="";
  ifname=ofname=btype="";
  btype="hbond"; // default
  npad=-1; lcut=hcut=dt=-1.;
  trajname="none"; // default value
  frm_ini=0; frm_fin=-1; stride=1; // default


```

```
  while (!ff.eof()) {
    ff>> sdum1;
    if (sdum1.find('#')==0) { // !=std::string::npos) {
      ff.getline(cdum,256); // comment. ignore the rest
    }
    else if (strcmp(sdum1.c_str(),"ifname")==0) ff>>ifname;
    else if (strcmp(sdum1.c_str(),"ofname")==0) ff>>ofname;
    else if (strcmp(sdum1.c_str(),"read_bond_data")==0) ff>>trajname;
    else if (strcmp(sdum1.c_str(),"btype")==0) ff>>btype;
    else if (strcmp(sdum1.c_str(),"npad")==0) ff>>npad;
    else if (strcmp(sdum1.c_str(),"lcut")==0) ff>>lcut;
    else if (strcmp(sdum1.c_str(),"hcut")==0) ff>>hcut;
    else if (strcmp(sdum1.c_str(),"segname")==0) ff>>segname;
    else if (strcmp(sdum1.c_str(),"dt")==0) ff>>dt;
    else if (strcmp(sdum1.c_str(),"frm_ini")==0) ff>>frm_ini;
    else if (strcmp(sdum1.c_str(),"frm_fin")==0) ff>>frm_fin;
    else if (strcmp(sdum1.c_str(),"stride")==0) ff>>stride;
    else {
      cout<< sdum1<<": Unrecognized option in " << cfname<<endl;
    }
  }
  if ((strlen(ifname.c_str())==0)&&(trajname=="none")) {
    cout<<"ERROR: No ifname in "<<cfname<<endl; exit(-1);
  }
  if  (strlen(ofname.c_str())==0) {
    cout<<"ERROR: No ofname in "<<cfname<<endl; exit(-1);
  }
  if  (strlen(segname.c_str())==0) {
    cout<<"ERROR: No segname in "<<cfname<<endl; exit(-1);
  }
//  if (npad==-1) { cout<<"ERROR: No npad in "<<cfname<<endl; exit(-1);}
//  if (lcut<0.) { cout<<"ERROR: No lcut in "<<cfname<<endl; exit(-1);}
//  if (hcut<0.) { cout<<"ERROR: No hcut in "<<cfname<<endl; exit(-1);}
  if ((dt<0.)&&(trajname=="none")) { cout<<"ERROR: No dt in "<<cfname<<endl; exit(-1);}
  hcut0=0.5*hcut;
  return;
}

/*************************************************************************/
void contact::get_occupancy()
{ /* Calculate occupancy and lifetime */
  int i,j, nf0=(frm_fin-frm_ini)/stride;
  double rdum;
  for (i=0;i<nbond;i++) {
    rdum=0.;
    for (j=frm_ini;j<frm_fin;j+=stride) rdum+=(double)traj[i][j];
    rdum/=(double)nf0;
    occ_bond.insert(pair<double,int>(rdum,i));
    bond_occ.insert(pair<int,double>(i,rdum));
  }
}

/*************************************************************************/
void contact::get_occ_traj()
{ /* Find occupancy trajectory  */
  int i,j,npad0=npad/2;
  double occ;

  if (nframe < (2*npad)) {
    cout<< "Too small number of frames. Reduce npad in get_bond.cpp."<<endl;
    exit(-1);
  }
```

```
  cout <<"Calculating occupancy trajectory with npad= "<<npad<<endl;
  for (i=0;i<nbond;i++) {
    // first check if first and last npad frames indicate transition
    for (j=0;j<(nframe-npad);j++) {
      occ = calculate_occupancy(i,j,j+npad);
      occ_traj[i][j+npad0]=occ;
    }
  }
}

/*************************************************************************/
void contact::get_transition()
{ /* Find bond transition. Algorithm:
    1) Use occ_traj for operations below.
    2) Use occupancy for first and last npad/2 points and check for
       transitions:
    2a) occ<lcut, occ>hcut at beginning/end respectively: find bond formation
    2b) occ<hcut, occ>lcut at beginning/end respectively: find bond breakage.
    2c) Otherwise: No transition. skip to the next bond.
    3) Find transition time for 2a) and 2b):
      In the case of bond breakage, reverse occ_traj and save into otrj.
      3a) Find first frame t0 where occ>lcut. Transition time is between
          this time t0 and end time nframe0=nframe-npad/2
      3b) Find avg occ between t0 and nframe0, called occ2
      3c) Increase t from t0, and find first frame t1 where occ>(0.5*occ2)
      3d) Find avg & std of occ between t1 and nframe0, called occ2
      3e) In the case of bond breakage, t1=nframe0-1-t1
          store t1, occ1, std(occ1).
  */
  int i,j,k,npad0=npad/2, nframe0=nframe-npad0, t0,t1,t2,t3,flag;
  int flag_t; // flag for transition type
  double occ,occ0,occ1,occ2,sig0,sig1,sig2;
  double rdum1,rdum2,rdum3,rdum4, *otrj=new double[nframe];
  cout<<"Finding bond formation/breakage events."<<endl;
  for (i=0;i<nbond;i++) {
    getavg(&occ0, &sig0, &occ_traj[i][npad0], npad0);
    getavg(&occ1, &sig1, &occ_traj[i][nframe-npad], npad0);
    // skip if no clear transition
    if ((occ0<=hcut)&&(occ0>=lcut)) continue; // intermed begin
    if ((occ1<=hcut)&&(occ1>=lcut)) continue; // intermed end
    if ((occ0<lcut)&&(occ1<lcut)) continue; // both low occ
    if ((occ0>hcut)&&(occ1>hcut)) continue; // both high occ
    // initial criterion passed.
    if (occ0 <lcut) { // search bond formation
      assert(occ1 > hcut);
      for (j=0;j<nframe;j++) otrj[j]=occ_traj[i][j];
      flag_t=0;
    }
    else { // search bond breakage (reverse search)
      assert(occ0>hcut); assert(occ1<lcut);
      for (j=0;j<nframe;j++) otrj[nframe-j-1]=occ_traj[i][j];
      flag_t=1;
    } // else { // search bond breakage
    rdum1=(flag_t==0)?occ1:occ0; rdum1*=0.5;
    flag=0;
    for (j=npad0;j<nframe0;j++) { // 3a)
      if (otrj[j]>lcut) {t0=j; flag=1; break;}
    }
    assert(flag==1); flag=0;
    getavg(&occ2,&sig2,&otrj[t0],(nframe0-t0)); // 3b)
    rdum2=0.5*occ2;
    for (j=t0;j<nframe0;j++) { // 3c)
```

```
      if (otrj[j]>rdum2) { t1=j; flag=1; break;}
    }
    assert(flag==1);
    getavg(&occ2,&sig2,&otrj[t1],(nframe0-t1)); // 3d)
    j=(flag_t==0)?t1:(nframe-1-t1);
    transition.insert(pair<int,int>(j,i));
    occ2=(flag_t==0)?occ2:-1.*occ2; // neg occupancy for bond breakage
    trans_occ.insert(pair<int,double>(i,occ2));
    trans_occ_std.insert(pair<int,double>(i,sig2));
  } // for (i=0;i<nbond;i++) {
}

/*************************************************************************/
void contact::read_data()
{
  ifstream ff(ifname.c_str());
  string line,sdum,sdum1,sdum2,sdum3,sdum4, linedata[12];
  int i,j,idum,idum1,idum2,flag;
  size_t found2;
  char **traj_tmp=new char*[maxbond];
  for (i=0;i<maxbond;i++) {
    traj_tmp[i]=new char[maxframe];
    for (j=0;j<maxframe;j++) traj_tmp[i][j]=0;
  }

  while (!ff.eof()) {
    /* specify which phrase to use as a beginning of a new block in
       charmm output */
    if (strcmp(btype.c_str(),"hbond")==0)
      found2=line.find("I-atom");
    else if (strcmp(btype.c_str(),"nonpolar")==0)
      found2=line.find("MIN DISTANCE");
    else { cout<<"ERROR: Wrong bond type!"<<endl; exit(-1);}

    if (found2!=std::string::npos) {
      getline(ff,line);
      // read one more line in case of hbond
      if (strcmp(btype.c_str(),"hbond")==0) getline(ff,line);
      std::size_t found3=line.find(segname.c_str());
      while (found3!=std::string::npos) { // read in distance block
        stringstream ss(line);
        for (i=0;i<8;i++) ss>>linedata[i];
        if (strcmp(btype.c_str(),"hbond")==0) {
          sdum1=linedata[1]; sdum2=linedata[6]; // resname
          sdum3=linedata[0]; sdum4=linedata[5]; // segid
          idum1=atoi(linedata[2].c_str());  // resids for h-bonds
          idum2=atoi(linedata[7].c_str());
        }
        else if (strcmp(btype.c_str(),"nonpolar")==0) {
          for (i=8;i<12;i++) ss>>linedata[i];
          sdum1=linedata[2]; sdum2=linedata[8]; // resname
          sdum3=linedata[1]; sdum4=linedata[7]; // segid
          idum1=atoi(linedata[3].c_str());  // resids
          idum2=atoi(linedata[9].c_str());
        }
        if ((sdum3==sdum4)&&(idum1 == idum2)) { //skip the same resid pair
          getline(ff,line); found3=line.find(segname.c_str());
          continue;
        }
        else if ((sdum3==sdum4)&&(idum1>idum2)) { // order resi w/in same segi
          idum=idum1; sdum=sdum1;
          idum1=idum2; sdum1=sdum2; idum2=idum; sdum2=sdum;
```

```
311         }
312         else {}
313         flag = 0; // flag for existing bond
314         for (i=0;i<nbond;i++) {
315           if ((sdum3==segi1[i])&&(idum1 == resi1[i])) {
316             if ((sdum4==segi2[i])&&(idum2 == resi2[i])) {
317               // existing bond
318               if (traj_tmp[i][nframe] == 0) { // avoid double counting
319                 traj_tmp[i][nframe] = 1; // bond i formed at nframe
320               }
321               flag=1; break;
322             }
323           }
324         } // for (i=0;i<nbond;i++) {
325         if (flag == 0) { // new bond
326           segi1[nbond]=sdum3; resn1[nbond]=sdum1; resi1[nbond]=idum1;
327           segi2[nbond]=sdum4; resn2[nbond]=sdum2; resi2[nbond]=idum2;
328           traj_tmp[nbond][nframe]=1;
329           ++nbond;
330         } // if (flag == 0) { // new bond
331         getline(ff,line); found3=line.find(segname.c_str());
332       } //while (found3!=std::string::npos) {
333       ++nframe;
334       if (nframe%500==0) cout << nframe<<" frames read"<<endl;
335     } // if (found2!=std::string::npos) {
336     getline(ff,line);
337   }
338   cout << nframe<<" frames read in total."<<endl;
339
340   traj=new char*[nbond];
341   if (frm_fin==-1) frm_fin=nframe; // default
342   if (frm_ini<0) frm_ini=0;
343
344   for (i=0;i<nbond;i++) {
345     traj[i]=new char[frm_fin-frm_ini];
346     for (j=frm_ini;j<frm_fin;j++) traj[i][j-frm_ini]=traj_tmp[i][j];
347   }
348   nframe=frm_fin-frm_ini;
349
350 //  if ((tmode==1)||(tmode==2)) { // use half of the traj
351 //    for (i=0;i<nbond;i++) {
352 //      traj[i]=new char[nframe/2];
353 //      if (tmode==1) {idum1=0; idum2=nframe/2;}
354 //      else { idum1=nframe/2; idum2=nframe; } // tmode==2
355 //      for (j=idum1;j<idum2;j++)       traj[i][j-idum1]=traj_tmp[i][j];
356 //    }
357 //    nframe/=2;
358 //  } //  if ((tmode==1)||(tmode==2)) { // use half of the traj
359 //  else {
360 //    assert(tmode==0);
361 //    for (i=0;i<nbond;i++) {
362 //      traj[i]=new char[nframe];
363 //      for (j=0;j<nframe;j++) traj[i][j]=traj_tmp[i][j];
364 //    }
365 //  }
366
367   // initialize occ_traj
368   occ_traj=new double*[nbond];
369   for (i=0;i<nbond;i++) occ_traj[i]=new double[nframe];
370
371   for (i=0;i<maxbond;i++) delete[] traj_tmp[i];
372   delete [] traj_tmp;
```

```
};

/*************************************************************************/
void contact::read_header(ifstream& ff)
{ // read file header from ff
  string sdum;
  ff >> sdum>>sdum>>sdum>>ifname; // original input filename
  ff >> sdum >> sdum >> dt >> sdum >> sdum >> sdum >> sdum >> sdum;
  getline(ff,sdum);
  getline(ff,sdum);
  //ff >> sdum >> sdum >> dt >> sdum >> lcut >> sdum >> hcut;
  ff >> sdum >> sdum >> sdum >> sdum >> sdum >> nframe >> sdum;
  getline(ff,sdum); // read off prev line
  getline(ff,sdum);
  ff >> sdum >> sdum >> sdum >> sdum >> nbond;
}

/*************************************************************************/
void contact::read_bond_data()
{ // read trajectory from existing file
  int i,j, idum, nf0;
  double rdum;
  ifstream ff(trajname.c_str());
  string line,sdum,sdum1,sdum2, linedata[8];

  read_header(ff);
  traj=new char*[nbond];
  if (frm_fin==-1) frm_fin=nframe;

  for (i=0;i<nbond;i++) traj[i]=new char[nframe];

  // initialize occ_traj
  cout<< "Reading bond trajectory from "<<trajname<<endl;
  occ_traj=new double*[nbond];
  for (i=0;i<nbond;i++) occ_traj[i]=new double[nframe];
  getline(ff,line); // end of previous line
  getline(ff,line); // "# Bond list: "
  for (i=0;i<nbond;i++) {
    ff >> sdum >> sdum >> segi1[i] >> resn1[i] >> resi1[i]
       >> segi2[i] >> resn2[i] >> resi2[i];
  }
  getline(ff,line); // end of previous line
  getline(ff,line); // blank line
  getline(ff,line); // "#frame ..."
  for (j=0;j<nframe;j++) { // read all frames
    getline(ff,sdum);
    stringstream ss(sdum);
    ss >> rdum; // read time
    for (i=0;i<nbond;i++) { ss >> idum; traj[i][j]=(char)idum;}
    //for (i=0;i<nbond;i++) { ss >> idum; traj[i][j-frm_ini]=(char)idum;}
  }
}

/*************************************************************************/
void contact::write_data()
{
  string ofn=ofname+".dat";
  ofstream ff(ofn.c_str());
  double t;
  int i,j,idum=0;
  idum=frm_ini; // if (tmode==2) idum=nframe;

```

```
  cout<< "Writing trajectory to "<<ofn<<endl;
  write_header(ff);
  ff << "# Bond list: "<<endl;
  for (i=0;i<nbond;i++) {
    ff << "# "<<setw(4)<<i<<"  "<<setw(4)<<segi1[i]<<" "
       <<setw(3)<<resn1[i]<<" "<<setw(4)<<setfill('0')<<resi1[i] <<" "
       <<setfill(' ')
       <<setw(4)<<segi2[i]<<" "<<setw(3)<<resn2[i]<<" "
       <<setw(4)<<setfill('0')<<resi2[i]<<setfill(' ')<<endl;
  }
  ff <<endl<<"#frame ";
  for (i=0;i<nbond;i++) ff <<setw(2)<<i <<setw(1)<<" ";
  ff<<endl;
  for (i=0;i<nframe;i++) {
    t= (double)(i+idum)*(double)dt;
    ff <<fixed<<setw(9)<<setprecision(3)<<t<<" ";
    for (j=0;j<nbond;j++) {
      ff<<setw(2)<< (int)traj[j][i]<<setw(1)<<" ";
    }
    ff << endl;
  }
}

/*****************************************************************************/
void contact::write_header(ofstream& ff)
{
  if (trajname=="none") ff << "# input filename: "  << ifname <<endl;
  else ff << "# input bond data: "  << trajname <<endl;
  ff << "# dt(ns)= "<< setw(5)<<setprecision(3)<< dt
     << "   npad= "<< setw(6)<< npad
     << "   lcut= "<< setw(5)<<setprecision(3)<< lcut
     << "   hcut= "<< setw(5)<<setprecision(3)<< hcut
     <<endl;
  ff<<"# npad<0: no local occ traj, [lh]cut<0.: no transition calculated."
    <<endl;

  ff << "# number of frames read: "  << nframe <<" "<<setw(8)
     <<dt*(double)nframe << "ns"<<endl;
  ff << "# frame range for analysis:  "<<setw(6)<<frm_ini<<" "
     << setw(6) << frm_fin<<endl;
  ff << "# number of bonds: " << nbond <<endl;
}

/*****************************************************************************/
void contact::write_occupancy()
{
  string ofn3=ofname+"_occ.dat";
  ofstream ff3(ofn3.c_str());
  int b, iframe, i;
  double occ,sig,rdum;
  multimap<double,int>::reverse_iterator rt;
  int idum=0;
  idum=frm_ini;

  cout << "Writing bond occupancy to "<<ofn3<<endl;
  write_header(ff3);
  ff3 << "# ind: index, bnum: bond number" <<endl;
  ff3 << "# Bond type and occupancy (ordered w/ occupancy): "<<endl;
  for (rt=occ_bond.rbegin();rt!=occ_bond.rend();rt++) {
    i=rt->second; rdum=rt->first;
    //ff3 << "# "<<setw(4)<<i<<"  "<<setw(4)<<segi1[i]<<" "
    ff3 << setw(4)<<i<<"  "<<setw(4)<<segi1[i]<<" "
```

```
        <<setw(3)<<resn1[i]<<" "<<setw(4)<<setfill('0')<<resi1[i] <<" "
        <<setfill(' ')
        <<setw(4)<<segi2[i]<<" "<<setw(3)<<resn2[i]<<" "
        <<setw(4)<<setfill('0')<<resi2[i]<<" "
        <<setfill(' ')<<setw(8)<<setprecision(5)<<fixed<<rdum<<endl;
  }

}

/*************************************************************************/
void contact::write_occ_traj()
{
  int npad0=npad/2;
  string ofn1=ofname+"_occ_traj.dat";
  ofstream ff(ofn1.c_str());
  double rdum,t;  int i,j;
  multimap<double,int>::reverse_iterator it;
  int idum=0;
  idum=frm_ini;

  cout<< "Writing occupancy trajectory to "<<ofn1<<endl;
  write_header(ff);
  ff << "# Bond type: "<<endl;
  for (i=0;i<nbond;i++) {
    ff << "# "<<setw(4)<<i<<"  "<<setw(4)<<segi1[i]<<" "
       <<setw(3)<<resn1[i]<<" "<<setw(4)<<setfill('0')<<resi1[i] <<" "
       <<setfill(' ')
       <<setw(4)<<segi2[i]<<" "<<setw(3)<<resn2[i]<<" "
       <<setw(4)<<setfill('0')<<resi2[i]<<setfill(' ')<<endl;
  }
  ff <<endl<<"#frame ";
  for (i=0;i<nbond;i++) ff <<setw(2)<<i <<setw(1)<<" ";
  ff<<endl;
  //  for (i=npad0;i<nframe-npad0;i++) {
  for (i=0;i<nframe;i++) {
    t= (double)(i+idum)*(double)dt;
    ff <<fixed<<setw(9)<<setprecision(3)<<t<<" ";
    for (j=0;j<nbond;j++) {
      ff<<setw(5)<< occ_traj[j][i]<<setw(1)<<" ";
    }
    ff << endl;
  }
}

/*************************************************************************/
void contact::write_transition()
{
  string ofn=ofname+".dat";
  string ofn1=ofname+"_break.dat",ofn2=ofname+"_form.dat";
  ofstream ff1(ofn1.c_str());  ofstream ff2(ofn2.c_str());
  multimap<int,int>::iterator it;
  multimap<int,double>::iterator jt,kt;
  int b, iframe, i,j;
  double occ,sig,rdum;
  int idum=0;

  cout << "Writing bond transition data to "<<ofname<<"_{break,form}.dat"<<endl;
  write_header(ff1);  write_header(ff2);
  ff1 << "# ind: index, bnum: bond number" <<endl;
  ff1 << "# occ,std(occ): occupancy and std while the bond is formed "<<endl;
  ff1 <<endl;
  ff1 << "# Broken bonds:"<<endl;
```

```
559     ff1<< "# ind   bnum                          t(frame)    t(ns)  occ    std(occ))"<<endl;
560
561     ff2 << "# ind: index, bnum: bond number "<<ofn <<endl;
562     ff2 << "# occ,std(occ): occupancy and std while the bond is formed "<<endl;
563     ff2 <<endl;
564     ff2 << "# Formed bonds:"<<endl;
565     ff2<< "# ind   bnum                          t(frame)    t(ns)  occ    std(occ))"<<endl;
566
567     i=j=0;
568     for (it=transition.begin();it!=transition.end();it++) {
569       iframe=it->first; b=it->second;
570       jt=trans_occ.find(b); occ=jt->second;
571       kt=trans_occ_std.find(b); sig=kt->second;
572       if (fabs(occ)<hcut) continue; // skip low-occupancy bonds
573       if (occ <0.) {
574         ff1 << "# "<<setw(4)<<i<<"  "<<setw(4)<<b<<" "
575             <<setw(4)<<segi1[b]<<" "
576             <<setw(3)<<resn1[b]<<" "<<setw(4)<<resi1[b] <<" "
577             <<setw(4)<<segi2[b]<<" "<<setw(3)<<resn2[b]<<" "
578             <<setw(4)<<resi2[b]
579             <<" " <<setw(6)
580             <<setfill('0')<<iframe<<setfill(' ')
581             << " " <<setw(9)<<setprecision(3)<<fixed<< dt*(double)iframe
582             <<" "<<setw(8)<<setprecision(5)<<fixed<< -1.0*occ
583             <<" "<<sig<<endl;
584         ++i;
585       }
586       else {
587         ff2 << "# "<<setw(4)<<i<<"  "<<setw(4)<<b<<" "
588             <<setw(4)<<segi1[b]<<" "
589             <<setw(3)<<resn1[b]<<" "<<setw(4)<<resi1[b] <<" "
590             <<setw(4)<<segi2[b]<<" "<<setw(3)<<resn2[b]<<" "
591             <<setw(4)<<resi2[b]
592             <<" "<<setw(6)
593             <<setfill('0')<<iframe<<setfill(' ')
594             << " " <<setw(9)<<setprecision(3)<<fixed<< dt*(double)iframe
595             <<" "<<setw(8)<<setprecision(5)<<fixed<< occ
596             <<" "<<sig<<endl;
597         ++j;
598       }
599     }
600
601 }
602
603 /*************************************************************************/
604 //void contact::write_transition()
605 //{
606 //  string ofn=ofname+".dat";
607 //  string ofn1=ofname+"_break.dat",ofn2=ofname+"_form.dat";
608 //  string ofn3=ofname+"_tot.dat";
609 //  ofstream ff1(ofn1.c_str());  ofstream ff2(ofn2.c_str());
610 //  ofstream ff3(ofn3.c_str());
611 //  multimap<int,int>::iterator it;
612 //  multimap<int,double>::iterator jt,kt;
613 //  int b, iframe, i,j;
614 //  double occ,sig,rdum;
615 //  multimap<double,int>::reverse_iterator rt;
616 //  int idum=0;
617 //  idum=frm_ini; // if (tmode==2) idum=nframe;
618 //
619 //  cout << "Writing bond transition data to "<<ofname
620 //       <<"_{break,form,tot}.dat"<<endl;
```

```
621 //  write_header(ff1);  write_header(ff2);
622 //  write_header(ff3);
623 //  ff1 << "# ind: index, bnum: bond number defined in "<<ofn <<endl;
624 //  ff1 << "# occ,std(occ): occupancy and std while the bond is formed "<<endl;
625 //  ff1 <<endl;
626 //  ff1 << "# Broken bonds:"<<endl;
627 //  ff1<< "# ind   bnum                          t(frame)    t(ns)  occ    std(occ))"<<endl;
628 //
629 //  ff2 << "# ind: index, bnum: bond number defined in "<<ofn <<endl;
630 //  ff2 << "# occ,std(occ): occupancy and std while the bond is formed "<<endl;
631 //  ff2 <<endl;
632 //  ff2 << "# Formed bonds:"<<endl;
633 //  ff2<< "# ind   bnum                          t(frame)    t(ns)  occ    std(occ))"<<endl;
634 //
635 //  ff3 << "# ind: index, bnum: bond number defined in "<<ofn <<endl;
636 //
637 //  i=j=0;
638 //  for (it=transition.begin();it!=transition.end();it++) {
639 //    iframe=it->first; b=it->second;
640 //    jt=trans_occ.find(b); occ=jt->second;
641 //    kt=trans_occ_std.find(b); sig=kt->second;
642 //    if (fabs(occ)<hcut) continue; // skip low-occupancy bonds
643 //    if (occ <0.) {
644 //      ff1 << "# "<<setw(4)<<i<<"  "<<setw(4)<<b<<" "
645 //        <<setw(4)<<segi1[b]<<" "
646 //        <<setw(3)<<resn1[b]<<" "<<setw(4)<<resi1[b] <<" "
647 //        <<setw(4)<<segi2[b]<<" "<<setw(3)<<resn2[b]<<" "
648 //        <<setw(4)<<resi2[b]
649 //        <<" " <<setw(6)
650 //        <<setfill('0')<<iframe<<setfill(' ')
651 //        << " " <<setw(9)<<setprecision(3)<<fixed<< dt*(double)(iframe+idum)
652 //        <<" "<<setw(8)<<setprecision(5)<<fixed<< -1.0*occ
653 //        <<" "<<sig<<endl;
654 //      ++i;
655 //    }
656 //    else {
657 //      ff2 << "# "<<setw(4)<<i<<"  "<<setw(4)<<b<<" "
658 //        <<setw(4)<<segi1[b]<<" "
659 //        <<setw(3)<<resn1[b]<<" "<<setw(4)<<resi1[b] <<" "
660 //        <<setw(4)<<segi2[b]<<" "<<setw(3)<<resn2[b]<<" "
661 //        <<setw(4)<<resi2[b]
662 //        <<" "<<setw(6)
663 //        <<setfill('0')<<iframe<<setfill(' ')
664 //        << " " <<setw(9)<<setprecision(3)<<fixed<< dt*(double)(iframe+idum)
665 //        <<" "<<setw(8)<<setprecision(5)<<fixed<< occ
666 //        <<" "<<sig<<endl;
667 //      ++j;
668 //    }
669 //  }
670 //
671 //  ff3 << "# Bond type and occupancy (ordered w/ occupancy): "<<endl;
672 //  for (rt=occ_bond.rbegin();rt!=occ_bond.rend();rt++) {
673 //    i=rt->second; rdum=rt->first;
674 //    ff3 << "# "<<setw(4)<<i<<"  "<<setw(4)<<segi1[i]<<" "
675 //      <<setw(3)<<resn1[i]<<" "<<setw(4)<<setfill('0')<<resi1[i] <<" "
676 //        <<setfill(' ')
677 //      <<setw(4)<<segi2[i]<<" "<<setw(3)<<resn2[i]<<" "
678 //      <<setw(4)<<setfill('0')<<resi2[i]<<" "
679 //      <<setfill(' ')<<setw(8)<<setprecision(5)<<fixed<<rdum<<endl;
680 //  }
681 //
682 //}
```

```
683
684 /*************************************************************************/
685 int main(int argc, char *argv[])
686 {
687   if (argc!=2) {
688     cout<<"Usage: te input.dat"<<endl; return -1;
689   }
690   int i,j,k;
691   string cfname=argv[1];
692
693   contact c1;
694   c1.get_args(cfname);
695   //cout<<c1.ifname <<" "<<c1.ofname<<" "<< c1.btype<<endl;
696   if (c1.trajname=="none") {
697     c1.read_data();   c1.write_data();
698   }
699   else c1.read_bond_data(); // read bond trajectory from existing file
700   c1.get_occupancy();
701   c1.write_occupancy();
702
703   if (c1.npad>0) {
704     c1.get_occ_traj();
705     c1.write_occ_traj();
706     if ((c1.hcut>0.)&&(c1.lcut>0.)) {
707       c1.get_transition();
708       c1.write_transition();
709     }
710   }
711   return 0;
712 }
713
714
715 /*************************************************************************/
716 void getavg(double *avg, double *sig, double ia[], int N)
717 /* Calculate average and s.d. of an array ia[] of size N. */
718 {
719   int i; double a, b, tol=-1.e-12;
720   a=b=0;
721   for (i=0;i<N;i++) { a+=ia[i]; b+=(ia[i]*ia[i]);}
722   a/=(double)N; b/=(double)N;
723   b=b-a*a;
724   if (b<0.) {assert(b>tol); b=0.;}
725   *avg=a; *sig=sqrt(b);
726   return;
727 }
728
729 /*************************************************************************/
730 void getminmax(double *min, int *imin, double *max, int *imax,
731      double ia[], int N)
732 /* Finds minimum & maximum values and their potisionos of array ia of size N */
733 {
734   int i, min0, max0;
735   double rmin,rmax;
736   min0=max0=0;
737   rmin=rmax=ia[0];
738   for (i=0;i<N;i++) {
739     if (ia[i]<rmin) {rmin=ia[i]; min0=i;}
740     if (ia[i]>rmax) {rmax=ia[i]; max0=i;}
741   }
742   *min=rmin; *max=rmax;
743   *imin=min0; *imax=max0;
744 }
```

APPENDIX X

GET CONTACT

The code gets the contacts with respect to thresholds [11].

```
# For get_contact.cpp
ifname stream_internal_hb_all.out
ofname hb_cabl
segname SH3
btype hbond
npad 40 # number of frames for initial/final padding
lcut 0.05 #  cutoff occupancy for no bond
hcut 0.5 #  cutoff occupancy for bond
dt 0.005 # coord saving frequency in ns.
```

APPENDIX XI

GET HBONDS

The code gets the hydrogen bonds with each atom [11].

```
* hb.inp: get hbonds
*

bomlev -1
stream ~/include/include.str

! set file names
set I 5cjb_neu ! input psf filename
set I1 5cjb_1   ! input dcd filename

read psf card name psf/@I.psf

!define katom sele (segi E000 .and. .not. ( type CA .or. type C )) end
define matom sele (segi OSCA .and. .not. type C* ) end
define catom sele ( (segi COL1 .or. segi COL2 .or. segi COL3) .and. .not. type C* ) end

prnlev 3 node 0

open read unit 11 file name dcd/@{I1}.dcd

traj query unit 11

traj firstu 11 nunit 1 begin ?START skip ?SKIP

!prnlev 2 node 0
  set j 1
  label L1
  traj read

!prnlev 5 node 0
  coor hbond sele matom end sele catom end verbose
!coor dist cut @{r0} sele katom end sele tatom end
!prnlev 3 node 0
  incr j by 1
if @j .le. ?nfile goto L1


stop

```

APPENDIX XII

CALCULATE RMSF

The code calculates the root mean squared fluctuations with each atom [11].

```
* coor_dyna.inp: calculate rmsd
*
bomlev -1
stream ~/include/include.str

! set file names
set I 5cjb_neu  ! input psf file
set I1 5cjb_1 ! input dcd file

read psf card name psf/@I.psf

open read unit 11 file name dcd/@{I1}.dcd

traj query unit 11
set start ?START
set skip ?SKIP

coor dyna sele ( segi OSCA .or. segi COL1 .or. segi COL2 .or. segi COL3 ) .and. type CA end NOPRINT
-
          firstu 11 nunit 1 begin @start skip @skip -
          orie SELE ( segi OSCA .or. segi COL1 .or. segi COL2 .or. segi COL3 ) .and. type CA END


scalar wmain show sele ( segi OSCA .or. segi COL1 .or. segi COL2 .or. segi COL3 ) .and. type CA end

stop

```

APPENDIX XIII

COLLAGEN DISTANCE BETWEEN ITS STRUCTURE

The code finds the distance between the alpha carbons and the collagen triple helix [12].

```
* distance.inp: get CA distance of collagen triple helical structure
*

bomlev -1
stream include/include.str

! set file names
set I 5cjb_neu ! input psf filename
set I1 5cjb_1  ! input dcd filename

read psf card name @I.psf

open read unit 11 file name @{I1}.dcd

traj query unit 11
set start ?START
set skip ?SKIP

!!!! residue number range in collagen
!!!! loop through collagen
set i 3      ! for COL3 - leading
calc j @i -1 ! for COL1 - middle
calc k @j -1 ! for COL2 - trailing
set imax 23  ! last C-terminal resid for COL3

label L0

! Correl facility
CORREL MAXT 5000000 MAXS 100
ENTER D1 dist COL3 @i CA COL1 @j CA ! L-M
ENTER D2 dist COL1 @j CA COL2 @k CA ! M-T
ENTER D3 dist COL2 @k CA COL3 @i CA ! T-L

! set traj
traj firstu 11 nunit 1 begin @start skip @skip

! write data
! number after trailing chain
open write card unit 91 name L-M_@k.dat
write D1 unit 91 dumb

open write card unit 92 name M-T_@k.dat
write D2 unit 92 dumb

open write card unit 93 name T-L_@k.dat
write D3 unit 93 dumb


END

incr i by 1
incr j by 1
incr k by 1

if @i .le. @{imax} goto L0

stop

```

APPENDIX XIV

CALCULATE CENTER OF MASS

The code calculates the center of mass for tubulin, beta sheets, and nucleotides [12].

```
* analdcd.inp: Calculate Center of Mass for Tubulin, Internal Beta Sheets, & Nucleotide
* revised by James on 05/21/19
*

!!! Settings !!!
!========================================================================!
bomlev -1
stream ~/include/include.str

! set file names
set I 5cjb_neu ! input psf filename
set I1 5cjb_1  ! input dcd filename

! read psf and dcd files
read psf card name psf/@I.psf
open read unit 11 file name dcd/@{I1}.dcd

traj query unit 11
set start ?START
set skip ?SKIP

prnlev 2 node 0

! set traj
traj firstu 11 nunit 1 begin @start skip @skip

!========================================================================!



!!! Measure Center of Mass
!========================================================================!

set m 1 ! for frames
label L0

traj read

! reset residue # at every new frame
!!!! residue number range in collagen
!!!! loop through collagen
set i 6  ! for COL3 - leading, skip the first three
calc j @i -1 ! for COL1 - middle
calc k @j -1 ! for COL2 - trailing
set imax 20 ! last C-terminal resid for COL3, skip the last three

! loop for triads/residue numbers
label L1

! aabb_: header for later awk

! Center of Mass for COL1/2/3
coor stat sele segi COL3 .and. resi @i .and. type CA end ! leading chain
set aabb_xl@{i} ?xave
set aabb_yl@{i} ?yave
set aabb_zl@{i} ?zave

coor stat sele segi COL1 .and. resi @j .and. type CA end ! middle chain
set aabb_xm@{i} ?xave
set aabb_ym@{i} ?yave
set aabb_zm@{i} ?zave

```

```
coor stat sele segi COL2 .and. resi @k .and. type CA end ! trailing chain
set aabb_xt@{i} ?xave
set aabb_yt@{i} ?yave
set aabb_zt@{i} ?zave

incr i by 1
incr j by 1
incr k by 1

if @i .le. @{imax} goto L1

prnlev 3 node 0

incr m by 1
if @{m} .le. ?nfile goto L0
!=====================================================================!



stop

```

APPENDIX XV

TRIAD TRAJECTORY FROM CENTER OF MASS

The code generates the triad trajectory from specific center of masses [12].

```
/* get_triad.cpp: Generate triad trajectory from specific center of mass
   coordinates of MT protofilament

   Usage: g++ get_triad_collagen.cpp -O3 -o gt_col
   ./gt_col [ifname] [ofname]
*/

#include <fstream>
#include <iostream>
#include <sstream>
#include <cassert>
#include <cstdio>
#include <iomanip>
#include <cmath>
#include <cstring>
#include <string>
#include <cstdlib>

using namespace std;

// Auxiliary Function Definitions
/*************************************************************************/
double dot(double *a, double *b);
void crossprod(double *v, double *w, double *c);
double normalize(double *a);
void writepsf(string ofname, int ntriad);
void writepdb(string ofname, int nframe, int ntriad, double ***cm,
              double ***t1, double ***t2, double ***t3);
/*************************************************************************/

int main(int argc, char **argv) {
  // Input Check and Error Catch
  /***********************************************************************/
  if (argc != 3) {
    cout << "Usage: ./gt [ifname] [ofname]" << endl;
    exit(-1);
  } // if (argc != 3) {
  /***********************************************************************/

  // Variable Declaration
  /***********************************************************************/
  bool flag = true;
  int i, j, k, ii, jj;
  int precision, threshold = 12; // Threshold for accuracy
  int maxframe = 2000; // Number of frames in simulation trajectory
  int ntriad = 15; // Number of triads per frame
  int ncrd = 135; // Number of coords per frame
  int nframe = 0;
  int o = 0, p = 1;
  int count = 1;
  int N = 10, subframe;

  double occ = 0.5;
  double r;
  double ***raw_crd;
  double **v;
  double ***v1, ***v2; // Nucleotide & Beta-sheet coordinates
  double ***cm, ***t1, ***t2, ***t3;
  double *a, *b;

  string ifname = argv[1];
  string ofname = argv[2];
```

```
  string sdum0, sdum1;
  /*************************************************************************/

  // Allocate Memory
  /*************************************************************************/
  raw_crd = new double**[maxframe]; v = new double*[maxframe];
  v1 = new double**[maxframe]; v2 = new double**[maxframe];
  cm = new double**[maxframe]; t1 = new double**[maxframe];
  t2 = new double**[maxframe]; t3 = new double**[maxframe];
  for (i = 0; i < maxframe; i++) {
    raw_crd[i] = new double*[ncrd]; v[i] = new double[ncrd];
    v1[i] = new double*[ntriad]; v2[i] = new double*[ntriad];
    cm[i] = new double*[ntriad]; t1[i] = new double*[ntriad];
    t2[i] = new double*[ntriad]; t3[i] = new double*[ntriad];
    for (j = 0; j < ncrd; j++) raw_crd[i][j] = new double[3];
    for (j = 0; j < ntriad; j++) {
      v1[i][j] = new double[3]; v2[i][j] = new double[3];
      cm[i][j] = new double[3]; t1[i][j] = new double[3];
      t2[i][j] = new double[3]; t3[i][j] = new double[3];
    } // for (j = 0; j < ntriad; j++) {
  } // for (i = 0; i < maxframe; i++) {
  a = new double[3]; b = new double[3];
  /*************************************************************************/

  // Read Data & Assign Raw Triads
  /*************************************************************************/
  cout << endl;
  cout << "Input read successfully." << endl;
  cout << " Using: " << ifname << " for input" << endl;
  cout << " Using: " << ofname << ".{dat}{psf}{pdb} for output" << endl;
  cout << endl;
  cout << "Reading data & assigning raw triads..." << endl;
  ifstream fin0(ifname);
  while (nframe < maxframe) {
    for (i = 0; i < ncrd; i++) {
      fin0 >> sdum0;
      fin0 >> sdum1;
      v[nframe][i] = stod(sdum1);
    } // for (i = 0; i < ncrd; i++) {
    for (j = 0; j < ntriad; j++) {
      for (k = 0; k < 3; k++) {
        cm[nframe][j][k] = (v[nframe][9*j + k] + v[nframe][9*j + 3 + k] + v[nframe][9*j + 6 +
k])/3 ; //center of a,b,c (o)
        v1[nframe][j][k] =(v[nframe][9*j + k] + v[nframe][9*j + 3 + k])/2 ;  // center of a,b (d)
        v2[nframe][j][k] = v[nframe][9*j + k] - v[nframe][9*j + 3 + k]; // vector b to a
      } // for (k = 0; k < 3; k++) {

      // Get t1, o -> d
      for (k = 0; k < 3; k++) {
        t1[nframe][j][k] = v1[nframe][j][k] - cm[nframe][j][k];
      } // for (k = 0; k < 3; k++) {

      // Normalize t1
      normalize(t1[nframe][j]);

      // Get t3
      crossprod(t1[nframe][j],v2[nframe][j],t3[nframe][j]);

      // Normalize t3
      normalize(t3[nframe][j]);

//      //r = abs(dot(v2[nframe][j], t3[nframe][j]));
```

```
124 //      r = dot(v2[nframe][j], t3[nframe][j]);
125
126       // Get t2
127       crossprod(t3[nframe][j], t1[nframe][j], t2[nframe][j]);
128
129       // Normalize t2
130       normalize(t2[nframe][j]);
131     } // for (j = 0; j < ntriad; j++) {
132     nframe++;
133   } // while (nframe < maxframe) {
134   cout << " " << nframe << " frames were read" << endl;
135   /*************************************************************************/
136
137   // Orthonormality Check
138   /*************************************************************************/
139   cout << endl;
140   cout << "Checking orthonormality of generated triads..." << endl;
141   threshold *= -1; precision = threshold - 1000;
142   while (precision <= threshold) {
143     for (i = 0; i < maxframe; i++) {
144       for (j = 0; j < ntriad; j++) {
145         if (dot(t1[i][j], t2[i][j]) >= 1.0*pow(10, precision)) flag = true;
146         else if (dot(t1[i][j], t3[i][j]) >= 1.0*pow(10, precision)) flag = true;
147         else if (dot(t2[i][j], t3[i][j]) >= 1.0*pow(10, precision)) flag = true;
148         else flag = false;
149         if (flag == true) break;
150       } // for (j = 0; j < ntriad; j++) {
151       if (flag == true) break;
152     } // for (i = 0; i < maxframe; i++) {
153     if (flag == false) break;
154     precision++;
155   } // while (precision <= threshold) {
156   if (precision > threshold) {
157     cout << " ERROR: TRIADS DO NOT MAINTAIN ORTHONORMALITY AT THE ACCURACY "
158          << "SPECIFIED THRESHOLD" << endl; exit(-1);
159   } // if (precision > threshold) {
160   cout << " Triads are orthonormal up to 1.0*10^(" << precision
161        << ") decimal places." << endl;
162   /*************************************************************************/
163
164   // Write Raw Data File
165   /*************************************************************************/
166   cout << endl;
167   cout << "Writing data file to be used as input for C++ triad analysis "
168        << "code..." << endl;
169   ofstream fout0(ofname + ".dat");
170   for (i = 0; i < maxframe; i++) {
171     for (j = 0; j < ntriad; j++) {
172       for (k = 0; k < 3; k++) {
173         fout0 << fixed << setprecision(8) << t1[i][j][k] << " ";
174       } // for (k = 0; k < 3; k++) {
175       fout0 << endl;
176       for (k = 0; k < 3; k++) {
177         fout0 << fixed << setprecision(8) << t2[i][j][k] << " ";
178       } // for (k = 0; k < 3; k++) {
179       fout0 << endl;
180       for (k = 0; k < 3; k++) {
181         fout0 << fixed << setprecision(8) << t3[i][j][k] << " ";
182       } // for (k = 0; k < 3; k++) {
183       fout0 << endl;
184       for (k = 0; k < 3; k++) {
185         fout0 << fixed << setprecision(8) << cm[i][j][k] << " ";
```

```
      } // for (k = 0; k < 3; k++) {
      fout0 << endl;
    } // for (j = 0; j < ntriad; j++) {
  } // for (i = 0; i < maxframe; i++) {
  cout << " Data written to: " << ofname << ".dat" << endl;
  /*************************************************************************/

  // Subsample Data In Overlapping Groups of 20,000 frames
  /*************************************************************************/
  cout << endl;
  cout << "Subsampling data in overlapping groups of 20,000 frames..." << endl;
  ofstream foutsub0("sub/" + ofname + "_0-200.dat");
  ofstream foutsub1("sub/" + ofname + "_100-300.dat");
  ofstream foutsub2("sub/" + ofname + "_200-400.dat");
  ofstream foutsub3("sub/" + ofname + "_300-500.dat");
  ofstream foutsub4("sub/" + ofname + "_400-600.dat");
  for (i = 0; i < maxframe; i++) {
    if (i < 20000) {
      for (j = 0; j < ntriad; j++) {
        for (k = 0; k < 3; k++) {
          foutsub0 << fixed << setprecision(8) << t1[i][j][k] << " ";
        } // for (k = 0; k < 3; k++) {
        foutsub0 << endl;
        for (k = 0; k < 3; k++) {
          foutsub0 << fixed << setprecision(8) << t2[i][j][k] << " ";
        } // for (k = 0; k < 3; k++) {
        foutsub0 << endl;
        for (k = 0; k < 3; k++) {
          foutsub0 << fixed << setprecision(8) << t3[i][j][k] << " ";
        } // for (k = 0; k < 3; k++) {
        foutsub0 << endl;
        for (k = 0; k < 3; k++) {
          foutsub0 << fixed << setprecision(8) << cm[i][j][k] << " ";
        } // for (k = 0; k < 3; k++) {
        foutsub0 << endl;
      } // for (j = 0; j < ntriad; j++) {
    } // if (i < 20000) {
    if (i >= 10000 && i < 30000) {
      for (j = 0; j < ntriad; j++) {
        for (k = 0; k < 3; k++) {
          foutsub1 << fixed << setprecision(8) << t1[i][j][k] << " ";
        } // for (k = 0; k < 3; k++) {
        foutsub1 << endl;
        for (k = 0; k < 3; k++) {
          foutsub1 << fixed << setprecision(8) << t2[i][j][k] << " ";
        } // for (k = 0; k < 3; k++) {
        foutsub1 << endl;
        for (k = 0; k < 3; k++) {
          foutsub1 << fixed << setprecision(8) << t3[i][j][k] << " ";
        } // for (k = 0; k < 3; k++) {
        foutsub1 << endl;
        for (k = 0; k < 3; k++) {
          foutsub1 << fixed << setprecision(8) << cm[i][j][k] << " ";
        } // for (k = 0; k < 3; k++) {
        foutsub1 << endl;
      } // for (j = 0; j < ntriad; j++) {
    } // if (i >= 10000 && i < 30000) {
    if (i >= 20000 && i < 40000) {
      for (j = 0; j < ntriad; j++) {
        for (k = 0; k < 3; k++) {
          foutsub2 << fixed << setprecision(8) << t1[i][j][k] << " ";
        } // for (k = 0; k < 3; k++) {
```

```
        foutsub2 << endl;
        for (k = 0; k < 3; k++) {
          foutsub2 << fixed << setprecision(8) << t2[i][j][k] << " ";
        } // for (k = 0; k < 3; k++) {
        foutsub2 << endl;
        for (k = 0; k < 3; k++) {
          foutsub2 << fixed << setprecision(8) << t3[i][j][k] << " ";
        } // for (k = 0; k < 3; k++) {
        foutsub2 << endl;
        for (k = 0; k < 3; k++) {
          foutsub2 << fixed << setprecision(8) << cm[i][j][k] << " ";
        } // for (k = 0; k < 3; k++) {
        foutsub2 << endl;
      } // for (j = 0; j < ntriad; j++) {
    } // if (i >= 20000 && i < 40000) {
    if (i >= 30000 && i < 50000) {
      for (j = 0; j < ntriad; j++) {
        for (k = 0; k < 3; k++) {
          foutsub3 << fixed << setprecision(8) << t1[i][j][k] << " ";
        } // for (k = 0; k < 3; k++) {
        foutsub3 << endl;
        for (k = 0; k < 3; k++) {
          foutsub3 << fixed << setprecision(8) << t2[i][j][k] << " ";
        } // for (k = 0; k < 3; k++) {
        foutsub3 << endl;
        for (k = 0; k < 3; k++) {
          foutsub3 << fixed << setprecision(8) << t3[i][j][k] << " ";
        } // for (k = 0; k < 3; k++) {
        foutsub3 << endl;
        for (k = 0; k < 3; k++) {
          foutsub3 << fixed << setprecision(8) << cm[i][j][k] << " ";
        } // for (k = 0; k < 3; k++) {
        foutsub3 << endl;
      } // for (j = 0; j < ntriad; j++) {
    } // if (i >= 30000 && i < 50000) {
    if (i >= 40000 && i < 60000) {
      for (j = 0; j < ntriad; j++) {
        for (k = 0; k < 3; k++) {
          foutsub4 << fixed << setprecision(8) << t1[i][j][k] << " ";
        } // for (k = 0; k < 3; k++) {
        foutsub4 << endl;
        for (k = 0; k < 3; k++) {
          foutsub4 << fixed << setprecision(8) << t2[i][j][k] << " ";
        } // for (k = 0; k < 3; k++) {
        foutsub4 << endl;
        for (k = 0; k < 3; k++) {
          foutsub4 << fixed << setprecision(8) << t3[i][j][k] << " ";
        } // for (k = 0; k < 3; k++) {
        foutsub4 << endl;
        for (k = 0; k < 3; k++) {
          foutsub4 << fixed << setprecision(8) << cm[i][j][k] << " ";
        } // for (k = 0; k < 3; k++) {
        foutsub4 << endl;
      } // for (j = 0; j < ntriad; j++) {
    } // if (i >= 40000 && i < 60000) {
  } // for (i = 0; i < maxframe; i++) {
  cout << " Subsampled data written to: sub/" << ofname << ".dat" << endl;
  /*************************************************************************/

  // Write Even & Odd Data Files
  /*************************************************************************/
  cout << endl;
```

```
  cout << "Writing even and odd data files..." << endl;
  ofstream foute(ofname + "_even.dat"); ofstream fouto(ofname + "_odd.dat");
  for (i = 0; i < maxframe; i++) {
    if (i%2 == 0) {
      for (j = 0; j < ntriad; j++) {
        for (k = 0; k < 3; k++) {
          foute << fixed << setprecision(8) << t1[i][j][k];
          if (k != 2) foute << " ";
        } // for (k = 0; k < 3; k++) {
        foute << endl;
        for (k = 0; k < 3; k++) {
          foute << fixed << setprecision(8) << t2[i][j][k];
          if (k != 2) foute << " ";
        } // for (k = 0; k < 3; k++) {
        foute << endl;
        for (k = 0; k < 3; k++) {
          foute << fixed << setprecision(8) << t3[i][j][k];
          if (k != 2) foute << " ";
        } // for (k = 0; k < 3; k++) {
        foute << endl;
        for (k = 0; k < 3; k++) {
          foute << fixed << setprecision(8) << cm[i][j][k];
          if (k != 2) foute << " ";
        } // for (k = 0; k < 3; k++) {
        foute << endl;
      } // for (j = 0; j < ntriad; j++) {
    } // if (i%2 == 0) {
    else {
      for (j = 0; j < ntriad; j++) {
        for (k = 0; k < 3; k++) {
          fouto << fixed << setprecision(8) << t1[i][j][k];
          if (k != 2) fouto << " ";
        } // for (k = 0; k < 3; k++) {
        fouto << endl;
        for (k = 0; k < 3; k++) {
          fouto << fixed << setprecision(8) << t2[i][j][k];
          if (k != 2) fouto << " ";
        } // for (k = 0; k < 3; k++) {
        fouto << endl;
        for (k = 0; k < 3; k++) {
          fouto << fixed << setprecision(8) << t3[i][j][k];
          if (k != 2) fouto << " ";
        } // for (k = 0; k < 3; k++) {
        fouto << endl;
        for (k = 0; k < 3; k++) {
          fouto << fixed << setprecision(8) << cm[i][j][k];
          if (k != 2) fouto << " ";
        } // for (k = 0; k < 3; k++) {
        fouto << endl;
      } // for (j = 0; j < ntriad; j++) {
    } // else {
  } // for (i = 0; i < maxframe; i++) {
  cout << " Data written to: " << ofname << "_{even}{odd}.dat" << endl;
  /*************************************************************************/

  // Split Total Trajectory Into 10 Separate Trajectories
  /*************************************************************************/
  cout << endl;
  cout << "Splitting single " << maxframe << " frame trajectory into 10, "
       << maxframe/10 << " frame trajectories..." << endl;
  ofstream fout1("split/" + ofname + "_1.dat");
  ofstream fout2("split/" + ofname + "_2.dat");
```

```
  ofstream fout3("split/" + ofname + "_3.dat");
  ofstream fout4("split/" + ofname + "_4.dat");
  ofstream fout5("split/" + ofname + "_5.dat");
  ofstream fout6("split/" + ofname + "_6.dat");
  ofstream fout7("split/" + ofname + "_7.dat");
  ofstream fout8("split/" + ofname + "_8.dat");
  ofstream fout9("split/" + ofname + "_9.dat");
  ofstream fout10("split/" + ofname + "_10.dat");
  for (i = 0; i < maxframe; i++) {
    if (count == 1) {
      for (j = 0; j < ntriad; j++) {
        for (k = 0; k < 3; k++) {
          fout1 << fixed << setprecision(8) << t1[i][j][k];
          if (k != 2) fout1 << " ";
        } // for (k = 0; k < 3; k++) {
        fout1 << endl;
        for (k = 0; k < 3; k++) {
          fout1 << fixed << setprecision(8) << t2[i][j][k];
          if (k != 2) fout1 << " ";
        } // for (k = 0; k < 3; k++) {
        fout1 << endl;
        for (k = 0; k < 3; k++) {
          fout1 << fixed << setprecision(8) << t3[i][j][k];
          if (k != 2) fout1 << " ";
        } // for (k = 0; k < 3; k++) {
        fout1 << endl;
        for (k = 0; k < 3; k++) {
          fout1 << fixed << setprecision(8) << cm[i][j][k];
          if (k != 2) fout1 << " ";
        } // for (k = 0; k < 3; k++) {
        fout1 << endl;
      } // for (j = 0; j < ntriad; j++) {
    } // if (count == 1) {
    else if (count == 2) {
      for (j = 0; j < ntriad; j++) {
        for (k = 0; k < 3; k++) {
          fout2 << fixed << setprecision(8) << t1[i][j][k];
          if (k != 2) fout2 << " ";
        } // for (k = 0; k < 3; k++) {
        fout2 << endl;
        for (k = 0; k < 3; k++) {
          fout2 << fixed << setprecision(8) << t2[i][j][k];
          if (k != 2) fout2 << " ";
        } // for (k = 0; k < 3; k++) {
        fout2 << endl;
        for (k = 0; k < 3; k++) {
          fout2 << fixed << setprecision(8) << t3[i][j][k];
          if (k != 2) fout2 << " ";
        } // for (k = 0; k < 3; k++) {
        fout2 << endl;
        for (k = 0; k < 3; k++) {
          fout2 << fixed << setprecision(8) << cm[i][j][k];
          if (k != 2) fout2 << " ";
        } // for (k = 0; k < 3; k++) {
        fout2 << endl;
      } // for (j = 0; j < ntriad; j++) {
    } // else if (count == 2) {
    else if (count == 3) {
      for (j = 0; j < ntriad; j++) {
        for (k = 0; k < 3; k++) {
          fout3 << fixed << setprecision(8) << t1[i][j][k];
          if (k != 2) fout3 << " ";
```

```
434         } // for (k = 0; k < 3; k++) {
435         fout3 << endl;
436         for (k = 0; k < 3; k++) {
437           fout3 << fixed << setprecision(8) << t2[i][j][k];
438           if (k != 2) fout3 << " ";
439         } // for (k = 0; k < 3; k++) {
440         fout3 << endl;
441         for (k = 0; k < 3; k++) {
442           fout3 << fixed << setprecision(8) << t3[i][j][k];
443           if (k != 2) fout3 << " ";
444         } // for (k = 0; k < 3; k++) {
445         fout3 << endl;
446         for (k = 0; k < 3; k++) {
447           fout3 << fixed << setprecision(8) << cm[i][j][k];
448           if (k != 2) fout3 << " ";
449         } // for (k = 0; k < 3; k++) {
450         fout3 << endl;
451       } // for (j = 0; j < ntriad; j++) {
452     } // else if (count == 3) {
453     else if (count == 4) {
454       for (j = 0; j < ntriad; j++) {
455         for (k = 0; k < 3; k++) {
456           fout4 << fixed << setprecision(8) << t1[i][j][k];
457           if (k != 2) fout4 << " ";
458         } // for (k = 0; k < 3; k++) {
459         fout4 << endl;
460         for (k = 0; k < 3; k++) {
461           fout4 << fixed << setprecision(8) << t2[i][j][k];
462           if (k != 2) fout4 << " ";
463         } // for (k = 0; k < 3; k++) {
464         fout4 << endl;
465         for (k = 0; k < 3; k++) {
466           fout4 << fixed << setprecision(8) << t3[i][j][k];
467           if (k != 2) fout4 << " ";
468         } // for (k = 0; k < 3; k++) {
469         fout4 << endl;
470         for (k = 0; k < 3; k++) {
471           fout4 << fixed << setprecision(8) << cm[i][j][k];
472           if (k != 2) fout4 << " ";
473         } // for (k = 0; k < 3; k++) {
474         fout4 << endl;
475       } // for (j = 0; j < ntriad; j++) {
476     } // else if (count == 4) {
477     else if (count == 5) {
478       for (j = 0; j < ntriad; j++) {
479         for (k = 0; k < 3; k++) {
480           fout5 << fixed << setprecision(8) << t1[i][j][k];
481           if (k != 2) fout5 << " ";
482         } // for (k = 0; k < 3; k++) {
483         fout5 << endl;
484         for (k = 0; k < 3; k++) {
485           fout5 << fixed << setprecision(8) << t2[i][j][k];
486           if (k != 2) fout5 << " ";
487         } // for (k = 0; k < 3; k++) {
488         fout5 << endl;
489         for (k = 0; k < 3; k++) {
490           fout5 << fixed << setprecision(8) << t3[i][j][k];
491           if (k != 2) fout5 << " ";
492         } // for (k = 0; k < 3; k++) {
493         fout5 << endl;
494         for (k = 0; k < 3; k++) {
495           fout5 << fixed << setprecision(8) << cm[i][j][k];
```

```
          if (k != 2) fout5 << " ";
        } // for (k = 0; k < 3; k++) {
        fout5 << endl;
      } // for (j = 0; j < ntriad; j++) {
    } // else if (count == 5) {
    else if (count == 6) {
      for (j = 0; j < ntriad; j++) {
        for (k = 0; k < 3; k++) {
          fout6 << fixed << setprecision(8) << t1[i][j][k];
          if (k != 2) fout6 << " ";
        } // for (k = 0; k < 3; k++) {
        fout6 << endl;
        for (k = 0; k < 3; k++) {
          fout6 << fixed << setprecision(8) << t2[i][j][k];
          if (k != 2) fout6 << " ";
        } // for (k = 0; k < 3; k++) {
        fout6 << endl;
        for (k = 0; k < 3; k++) {
          fout6 << fixed << setprecision(8) << t3[i][j][k];
          if (k != 2) fout6 << " ";
        } // for (k = 0; k < 3; k++) {
        fout6 << endl;
        for (k = 0; k < 3; k++) {
          fout6 << fixed << setprecision(8) << cm[i][j][k];
          if (k != 2) fout6 << " ";
        } // for (k = 0; k < 3; k++) {
        fout6 << endl;
      } // for (j = 0; j < ntriad; j++) {
    } // else if (count == 6) {
    else if (count == 7) {
      for (j = 0; j < ntriad; j++) {
        for (k = 0; k < 3; k++) {
          fout7 << fixed << setprecision(8) << t1[i][j][k];
          if (k != 2) fout7 << " ";
        } // for (k = 0; k < 3; k++) {
        fout7 << endl;
        for (k = 0; k < 3; k++) {
          fout7 << fixed << setprecision(8) << t2[i][j][k];
          if (k != 2) fout7 << " ";
        } // for (k = 0; k < 3; k++) {
        fout7 << endl;
        for (k = 0; k < 3; k++) {
          fout7 << fixed << setprecision(8) << t3[i][j][k];
          if (k != 2) fout7 << " ";
        } // for (k = 0; k < 3; k++) {
        fout7 << endl;
        for (k = 0; k < 3; k++) {
          fout7 << fixed << setprecision(8) << cm[i][j][k];
          if (k != 2) fout7 << " ";
        } // for (k = 0; k < 3; k++) {
        fout7 << endl;
      } // for (j = 0; j < ntriad; j++) {
    } // else if (count == 7) {
    else if (count == 8) {
      for (j = 0; j < ntriad; j++) {
        for (k = 0; k < 3; k++) {
          fout8 << fixed << setprecision(8) << t1[i][j][k];
          if (k != 2) fout8 << " ";
        } // for (k = 0; k < 3; k++) {
        fout8 << endl;
        for (k = 0; k < 3; k++) {
          fout8 << fixed << setprecision(8) << t2[i][j][k];
```

```
          if (k != 2) fout8 << " ";
        } // for (k = 0; k < 3; k++) {
        fout8 << endl;
        for (k = 0; k < 3; k++) {
          fout8 << fixed << setprecision(8) << t3[i][j][k];
          if (k != 2) fout8 << " ";
        } // for (k = 0; k < 3; k++) {
        fout8 << endl;
        for (k = 0; k < 3; k++) {
          fout8 << fixed << setprecision(8) << cm[i][j][k];
          if (k != 2) fout8 << " ";
        } // for (k = 0; k < 3; k++) {
        fout8 << endl;
      } // for (j = 0; j < ntriad; j++) {
    } // else if (count == 8) {
    else if (count == 9) {
      for (j = 0; j < ntriad; j++) {
        for (k = 0; k < 3; k++) {
          fout9 << fixed << setprecision(8) << t1[i][j][k];
          if (k != 2) fout9 << " ";
        } // for (k = 0; k < 3; k++) {
        fout9 << endl;
        for (k = 0; k < 3; k++) {
          fout9 << fixed << setprecision(8) << t2[i][j][k];
          if (k != 2) fout9 << " ";
        } // for (k = 0; k < 3; k++) {
        fout9 << endl;
        for (k = 0; k < 3; k++) {
          fout9 << fixed << setprecision(8) << t3[i][j][k];
          if (k != 2) fout9 << " ";
        } // for (k = 0; k < 3; k++) {
        fout9 << endl;
        for (k = 0; k < 3; k++) {
          fout9 << fixed << setprecision(8) << cm[i][j][k];
          if (k != 2) fout9 << " ";
        } // for (k = 0; k < 3; k++) {
        fout9 << endl;
      } // for (j = 0; j < ntriad; j++) {
    } // else if (count == 9) {
    else if (count == 10) {
      for (j = 0; j < ntriad; j++) {
        for (k = 0; k < 3; k++) {
          fout10 << fixed << setprecision(8) << t1[i][j][k];
          if (k != 2) fout10 << " ";
        } // for (k = 0; k < 3; k++) {
        fout10 << endl;
        for (k = 0; k < 3; k++) {
          fout10 << fixed << setprecision(8) << t2[i][j][k];
          if (k != 2) fout10 << " ";
        } // for (k = 0; k < 3; k++) {
        fout10 << endl;
        for (k = 0; k < 3; k++) {
          fout10 << fixed << setprecision(8) << t3[i][j][k];
          if (k != 2) fout10 << " ";
        } // for (k = 0; k < 3; k++) {
        fout10 << endl;
        for (k = 0; k < 3; k++) {
          fout10 << fixed << setprecision(8) << cm[i][j][k];
          if (k != 2) fout10 << " ";
        } // for (k = 0; k < 3; k++) {
        fout10 << endl;
      } // for (j = 0; j < ntriad; j++) {
```

```
    } // else if (count == 10) {
    count++;
    if (count == 11) count = 1;
  } // for (i = 0; i < maxframe; i++) {
  cout << " Split trajectories written to files: split/" << ofname
       << "_{0 ... 10}.dat" << endl;
  /*************************************************************************/

  // Split Total Trajectory Into N Separate Trajectories
  /*************************************************************************
  cout << endl;
  cout << "Splitting total trajectory into " << N << " separate "
       << "trajectories..." << endl;
  subframe = maxframe/N;
  for (i = 0; i < maxframe; i++) {
    for (j = 0; j < N; j++) {
      if (i%j == 0) {

      } //if (i%j == 0) {
    } // for (j = 0; j < N; j++) {
  } // for (i = 0; i < maxframe; i++) {
  cout << " " << maxframe << " trajectory split into " << N << ", "
       << subframe << " trajectories, written to files in: split/" << ofname
       << "_{0..." << N << "}.dat" << endl;
  *************************************************************************/

  // Write PSF & PDB Files for VMD Visualization
  /*************************************************************************/
  cout << endl;
  cout << "Writing PSF file for visualization in VMD..." << endl;
  writepsf(ofname, ntriad);
  cout << " Protein structure written to: " << ofname << ".psf" << endl;
  cout << endl;
  cout << "Writing PDB file for visualization in VMD..." << endl;
  writepdb(ofname, nframe, ntriad, cm, t1, t2, t3);
  cout << " Atom trajectories written to: " << ofname << ".pdb" << endl;
  /*************************************************************************/

  return 0;
}

// Auxiliary Functions
/*************************************************************************/
double dot(double *a, double *b) {
  double z = a[0]*b[0] + a[1]*b[1] + a[2]*b[2];
  return z;
}

void crossprod(double *v, double *w, double *c) {
  c[0] = v[1]*w[2] - v[2]*w[1];
  c[1] = v[2]*w[0] - v[0]*w[2];
  c[2] = v[0]*w[1] - v[1]*w[0];
}

double normalize(double *a) {
  int i; double rdum = 0.0;
  for (i = 0; i < 3; i++) rdum += (a[i]*a[i]);
  rdum = sqrt(rdum);
  for (i = 0; i < 3; i++) a[i] /= rdum;
  return rdum;
}
```

```
682 void writepsf(string ofname, int ntriad) {
683   int i, j, k;
684   int ii, jj;
685   int count;
686
687   ofstream fout(ofname + ".psf");
688   fout << "PSF CMAP CHEQ" << endl;
689   fout << endl;
690   fout << " !NTITLE" << endl;
691   fout << "*" << endl;
692   fout << setw(6) << 4*ntriad << " !NATOM" << endl;
693   for (i = 0; i < ntriad; i++) {
694     ii = 4*i + 1;
695     fout << setfill(' ') << setw(8) << ii << " t0000 " << setfill('0')
696          << setw(4) << i << "  TRI  O     0   0.000000        1.0000"
697          << "               0   0.00000      0.000000E-02" << endl;
698     for (j = 0; j < 3; j++) {
699       ii++;
700       fout << setfill(' ') << setw(8) << ii << " t0000 " << setfill('0')
701            << setw(4) << i << "  TRI  S     0   0.000000        1.0000"
702            << "               0   0.00000      0.000000E-02" << endl;
703     } // for (j = 0; j < 3; j++) {
704   } // for (i = 0; i < ntriad; i++) {
705   fout << endl;
706   fout << setfill(' ') << setw(8) << 3*ntriad << " !NBOND" << endl;
707   for (i = 0; i < ntriad; i++) {
708     ii = 4*i + 1;
709     for (j = 0; j < 3; j++) {
710       jj = ii + j + 1;
711       fout << setw(8) << ii << setw(8) << jj;
712       count++;
713       if (count%4 == 0) fout << endl;
714     } // for (j = 0; j < 3; j++) {
715   } // for (i = 0; i < ntriad; i++) {
716 }
717
718 void writepdb(string ofname, int nframe, int ntriad, double ***cm,
719               double ***t1, double ***t2, double ***t3) {
720   int i, j, k, l;
721   int ii, jj, kk;
722   double *a = new double[3];
723   double *b = new double[3];
724
725   ofstream fout(ofname + ".pdb");
726   for (i = 0; i < nframe; i++) {
727     fout << "MODEL" << endl;
728     for (j = 0; j < ntriad; j++) {
729       for (k = 0; k < 3; k++) a[k] = cm[i][j][k];
730       jj = 4*j + 1;
731 //ATOM      1  N   HIS A   1      49.668  24.248  10.436  1.00 25.00           N
732 //ATOM   1060  C   ARG A 141      -8.119  13.499  -9.393  6.00 28.93           C
733       fout << "ATOM" << "   " << setfill(' ') << setw(4) << jj << "  " << "O  "
734            << " " << "TRI" << " " << "A" << " " << setw(3) << j + 1 << "     "
735            << setw(7) << fixed << setprecision(3) << a[0] << " "
736            << setw(7) << fixed << setprecision(3) << a[1] << " "
737            << setw(7) << fixed << setprecision(3) << a[2] << " "
738            << " 0.50" << " " << " 0.00" << "      " << "t000" << " " << "O"
739            << endl;
740       for (k = 0; k < 3; k++) {
741         if (k == 0) for (l = 0; l < 3; l++) b[l] = 15*t1[i][j][l] + a[l];
742         if (k == 1) for (l = 0; l < 3; l++) b[l] = 15*t2[i][j][l] + a[l];
743         if (k == 2) for (l = 0; l < 3; l++) b[l] = 15*t3[i][j][l] + a[l];
```

```
        kk = jj + k + 1;
        fout << "ATOM" << "   " << setfill(' ') << setw(4) << jj << "  "
             << "S  " << " " << "TRI" << " " << "A" << " " << setw(3) << j + 1
             << "     "
             << setw(7) << fixed << setprecision(3) << b[0] << " "
             << setw(7) << fixed << setprecision(3) << b[1] << " "
             << setw(7) << fixed << setprecision(3) << b[2] << " "
             << " 0.50" << " " << " 0.00" << "      " << "t000" << " " << "S"
             << endl;
      } // for (k = 0; k < 3; k++) {
    } // for (j = 0; j < ntriad; j++) {
    fout << "ENDMDL" << endl;
  } // for (i = 0; i < nframe; i++) {
}
/*****************************************************************/
```