Effect of Explicit and Implicit Solvents in the Binding Free Energies of Protein Complexes

Katarina Albrechtas

Department of Chemistry

Western University

March 31st 2020

Submitted in partial fulfillment of the requirements for CHEM 4491E

Supervisor: Dr. Styliani Consta

Contents

Abstract

Solvation on biomolecules is an important subject because it is directly associated with life. For this reason there are a significant number of experimental and computational studies that aim at unravelling solvation. In computations, there is a significant amount of contradictory information regarding which solvation model is the best to use, which is currently demonstrated in the literature. There are many factors that need to be considered when determining which solvation model is best to use for a simulation in computational chemistry; for example, computational cost and chemical accuracy. The two models that are compared are the *explicit and implicit solvation models*. Each model offers its own array of distinct pros and cons. This thesis focuses on the effects of the two different solvation models mentioned above on the binding free energies of the proteins $\boldsymbol{\beta}$-Trypsin and Bovine pancreatic trypsin inhibitor. It is found that there is a significant difference in estimated binding free energies between the implicit and explicit model. However, since both also varied significantly from the experimental value, it is impossible to determine which model is more accurate.

Acknowledgements

I would like to thank Dr. Styliani Consta for all of her assistance and guidance throughout this fourth year thesis project. With her invaluable support I was able to acquire a great deal of knowledge in regard to computational chemistry and improve my research skills. I would also like to thank Victor and the rest of the Consta group for their support throughout this thesis project, without everyone’s support this would not have been possible.

List of Abbreviations

MD – Molecular Dynamics

GB – Generalized Born

TIP3P – Three-Site Transferable Intermolecular Potential

VMD – Visual Molecular Dynamics

NAMD – Nanoscale Molecular Dynamics

FEP - Free energy perturbation

EEF - Effective Energy Function

ABSINTH- Self-Assembly of Biomolecules Studied by an Implicit, Novel, and Tunable Hamiltonian

RDF – Radial Distribution Function

RMSD - Root-mean-square deviation of atomic positions

List of Figures

List of Tables

1 Introduction

Solvation of biomolecules is a critical problem in computational chemistry. Atomistic models of large biomolecules such as proteins, is important for understanding allosteric mechanisms, drug design, protein folding, protein-ligand binding.[4,26,27] Being able to accurately determine the solvation free energy of large biomolecules is of great interest to a variety of fields, including biomedical, chemical and industrial research[22,3]. However, accurately studying these large biomolecule models can be difficult due to their size. Simulating the behaviour of biomolecules requires a delicate balance between model detail and sampling efficiency[4].

Currently the most detailed way to conduct these simulations is to use an explicit model, which means all the solvent molecules are treated at the atomistic level. However, using an explicit model greatly affects the sampling efficiency, which is determined by the sufficient collection of statistically independent molecular states of the system. This is because there is a significant computational cost associated with simulating the large number of solvent molecules required to model a bulk solution[20] and the slow (in comparison with the simulation time) conformational changes of the macromolecules. Also, even with all the extra detail that goes into explicit modelling it is not exempt from approximations and error[20], because the accuracy of the explicit model depends on the limitations of the force field.

In an attempt to try to improve efficiency of these simulations, implicit solvent models have been devised. Implicit models treat the solvent as a continuum medium by coupling electrostatics with statistical mechanics via the Poisson-Boltzmann equation[10]. Implicit models are very useful in computational chemistry for several reasons. They eliminate the lengthy equilibration time of water seen in explicit models[15], improve sampling[15], and there are no artifacts due to periodic boundary conditions[15], and new simple ways to estimate free energies become possible. Although implicit models may reduce the computational expense associated with the simulations of macromolecules there are caveats, because implicit models also have disadvantages. These disadvantages include; lack of dynamic information about the rate of transitions between conformations of macromolecules, not all aspects of transition mechanisms can be captured because continuum solvent modelling eliminates water molecules that may be significant for protein-protein interactions, and lack of information about the entropy of the solvent. Several current studies compare the accuracy of different implicit models by comparing them to explicit model results. Shivakumar et al.[22] compared different implicit model results to explicit results of solvation free energies of small molecules to determine which implicit model agrees the most with the explicit model[22]. The problem with this approach is that there is an assumption that the explicit model is accurate, which has been disproven by some scientists. Saeed Izadi et al.[7] compared different explicit and implicit models and found large discrepancies between different types of explicit and implicit models in the calculation of Protein-Ligand electrostatic binding free energies. The purpose of their study was to find an accurate but efficient computational model so they evaluated the accuracy of the generalized born (GB) solvent model[7]. The r^2 correlation between the implicit models was lower then between the explicit models, demonstrating how important choosing the correct implicit model can be to determining the accuracy of results. The authors concludes that the only true way to determine if one of these models is accurate is to compare it to experimental results, which is also not straightforward and it is very challenging to do well. It is difficult to compare computational with experimental results for the binding free energies of protein complexes because these complexes are usually flexible and have may degrees of freedom which introduces large amounts of uncertainty.[7]

Comparing implicit and explicit solvation models is important so it can be determined if these models can truly be used interchangeably[9,23,24,25]. It is also important to determine the exact accuracy of each implicit solvation model and to compare them to experimentally obtained values to see if using an implicit solvation model is a viable way to determine solvation free energies. If it is determined that implicit solvent models are just as accurate in molecular dynamics simulations as explicit models, this could save researchers time and effort and more computational power could be spent on protein dynamics rather then on solvent modelling[4].

There are several methods of implicit modelling. For instance Cumberworth et al.[4] chose to compare the free energies of solvation using explicit and implicit solvent in the context of protein folding[1]. They chose to use eight different implicit models including: a finite difference Poisson Boltzmann model (PBEQ)[4], Generalized Born using Molecular Volume 2 (GBMV2)[4], Fast Analytical Continuum Treatment of Solvation (FACTS)[4], Analytical Continuum Solvent (ACE)[4], Screened Coulomb Potentials (SCP)[4], Effective Energy Function 1 (EEF1)[4,13], and Self-Assembly of Biomolecules Studied by an Implicit, Novel, and Tunable Hamiltonian (ABSINTH)[4]. The explicit solvent model they chose to employ was Three-Site Transferrable Intermolecular Potential (TIP3P)[4,8,14], which seemed to be the most popular explicit model used across the literature. In the end they found that most of the implicit free energy values were a lot lower then the reference and explicit values, having a difference of about 5 kcal/mol[4]. The three implicit models that had values closest to the free energy values for the explicit model were the effective energy functions (EEFs)[4,13], ABSINTH[4] and EEF1[4]. Scientists have also tried changing certain parameters to increase the accuracy of the implicit free energy values. However, none of these changes improved the implicit free energy values significantly[4]. Many other papers ran similar experiments trying different implicit and explicit models to determine which gives the most accurate results, and it seems that opinions and results vary from article to article except for TIP3P being the most widely used explicit model. Based on a number of articles it seems that the best implicit models would be the Effective Energy Function (EEF1)[4,12,13], Poisson-Boltzmann (PB)[18] or the Generalized Born model (GB) of solvation[1,2,7,10,16,22]. For the TIP3P explicit model an article by David L. Mobley[14] uses molecular dynamic free energy simulations with TIP3P explicit solvent to compute the hydration free energies of 504 neutral small organic molecules and compared the results to experiment values[14]. They found a good correlation (r^2 of 0.891 $\pm$ 0.06) and a root mean squared (rms) error of 1.24 $\pm$ 0.01 kcal/mol or roughly 2 kT (where k is Boltzmann constant and T temperature) and determined that the TIP3P model represents the accuracy that should be expected from the best current physical models for hydration free energies[14]. Based on these results seen in literature for this experiment a TIP3P model will be used for the explicit modelling and the GB method will be employed for the implicit model.

The motivation of my thesis is to determine the binding free energy of $\boldsymbol{\beta}$-Trypsin and Bovine pancreatic trypsin inhibitor using an implicit and explicit model and then to compare these values to each other and to experimental values. This protein complex has been selected because it has an already known experimentally determined association energy. Thus, the model provides a baseline value to compare my implicit and explicit values to. It is important to have a reference value because an improper assumption to make would be that the explicit solvent model is accurate and that it could be used as a reference to compare all my implicit values to[4].

The goal of my project is to determine if the explicit model is as accurate as it is seen in the literature, as well as to find out the level of accuracy the implicit models provide. In the literature it is seen that a desired chemical accuracy is approximately 1 kcal/mol, however this value varies depending on what the values are being used for[7]. The 1 kcal/mol refers to the standards expected from using these techniques in drug development[7]. If it is found that the implicit model I choose to employ has a high degree of accuracy, comparable with experimental and explicit binding free energy values, then this methods could be used moving forward when calculating the binding free energies of protein complexes to save future computational costs when running these simulations.

Before beginning this thesis, I wrote a molecular dynamics (MD) code in C++ to learn about each aspect of the larger code used in Nanoscale Molecular Dynamics (NAMD) to run my simulations. The code has been included in Appendix 1. My code ran molecular dynamics simulations with argon spheres, which is a much simpler system then the protein complex in solvents. This exercise taught me about the different aspects that are normally found within an MD code, including Lennard-Jones and force calculations, as well as the Verlet algorithm. A further explanation of MD can be found in the methodology.

2 Methodology

2.1 Molecular Dynamics

A molecular dynamic (MD) approach was used to study the physical movements of the atoms in the system being studied. Molecular dynamic methods allow for the study of the behaviour of systems of many interacting molecules[6,17,21,5]. Because of the complexity of the interactions it is impossible to determine these properties analytically for large systems, such as in this case protein complexes, which is why a MD method must be used. Molecular dynamic simulations work by solving Newton's equation of motion for all the molecules in the system using discretization in the positions, momenta and forces. The time is also discrete and for this reason there is a time step.

Newton's second law of motion is expressed as;

$$\vec{F_i} = -\nabla U = m_i \frac{d^2 \vec{r_i}}{dt^2} \tag{1}$$

where $\vec{F_i}$ is the sum of all forces acting on an atom *i*, m is the mass of an atom *i*, *t* is time, $\vec{r}$ is the position of the specific atom *i*, and U is the potential energy of the system. ∇ denotes the gradient and *U* is a function (scalar quantity) of all the atomic positions.

Since one cannot solve Eq. (1) analytically for many atoms, a numerical finite-difference method is used to integrate Eq. (1). An important finite difference method used in molecular dynamics simulation is the Verlet algorithm[17]. In the MD computer code I prepared (Appendix 1) I programmed the Verlet algorithm and for this reason it is discussed in the thesis. The Verlet algorithm arises from the Taylor series expansion of the position of an atom forward and back in time. The Verlet algorithm is expressed as;

$$\vec{r}\,(\mathrm{t}+\ \delta t\,) = 2\vec{r}(\mathrm{t}) - \vec{r}\,(t\ -\ \delta t\,) + \vec{a}(t)\delta t^2 + O(\delta t^4) \tag{2}$$

where $\vec{r}$ is the position, $\vec{a}$ is the acceleration (second derivative of the position of an atom with respect to time), δt is the time step, and O denotes order of magnitude of the error because of the truncation of the Taylor expansion. This local error because of the truncation of the Taylor expansion is of the order of the fourth power of the time step. The global error is worse, it is of the order of second power of the time step. The Verlet algorithm uses positions and accelerations at time t and positions from time t - δt to calculate new positions at time t + δt. A typical time step in classical simulations is 0.5 fs or 1.0 fs. The Equation 2 is applied to every single atomic site in the system at every time step. Thus, a computer code cycles through this algorithm a huge number of times since it repeats it for every atomic site and for millions of time steps. In addition to the Verlet algorithm there are other similar integration algorithms such as the velocity Verlet and the leap-frog. In the software NAMD that I used in my simulations the velocity Verlet algorithm is implemented. The velocity Verlet is similar to the Verlet algorithm but the local error due to the truncation of the Taylor expansion is of the order of the third power of the time step. The algorithm is expressed as

$$\vec{r}(t+\ \delta t) =\ \vec{r}(t) +\ \vec{v}(t)\delta t + \frac{1}{2}\vec{a}(t)\delta t^2 \tag{3}$$

$$\vec{v}(t+\ \delta t) =\ \vec{v}(t) +\ \frac{1}{2}[\vec{a}(t) +\ \vec{a}(t+\delta t)]\delta t \tag{4}$$

where $\vec{r}$ is the position, $\vec{v}$ is the velocity, $\vec{a}$ is the acceleration, t is time and δt is the time step.

2.2 Protein Complex

The protein complex that will be studied is Bovine pancreatic trypsin inhibitor binding with $\boldsymbol{\beta}$-Trypsin.

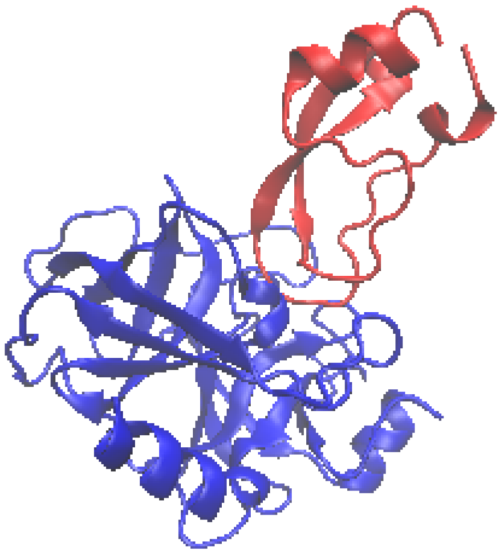

Figure 1: Typical snapshot of the protein complex, bovine pancreatic trypsin inhibitor (red) binding with $\boldsymbol{\beta}$-Trypsin (blue) structure generated using Visual Molecular Dynamics (VMD) software.[29]

This protein complex was chosen because in the literature this protein is a standard benchmark for assessing the quality of a force field and solvation models[11]. The reason this protein complex is suitable for atomistic computational modelling is because it follows the lock and key model[7], which means it is a rigid protein complex. Rigidity is important because the conformational changes that can be seen in other models such as induced fit[7] cannot be captured within the simulation time of atomistic modelling. In the literature it was found that the association energy for the interaction of $\boldsymbol{\beta}$-Trypsin and Bovine pancreatic trypsin inhibitor is 15 kcal/mol [11]. This value was determined experimentally and will be used as a benchmark to compare my values to, which will be determined computationally.

2.3 Force Field

The force field in the context of computer simulations is vital to describing the intra- and inter- molecular potential energies of a collection of atoms, and the corresponding parameters that will determine the energy of the configuration[17]. There are many different types of functional force fields, but the general equation that describes the main class of force fields can be written as;

$$U = \sum_{bonds} \frac{1}{2} k_r (r_{ij} - r_0)^2 + \sum_{angles} \frac{1}{2} k_\theta (\theta_{ijk} - \theta_0)^2 + \sum_{torsions} \sum_n k_{\phi,n} \left[\cos\left(n\phi_{ijk\ell} + \delta_n\right) + 1\right] + \sum_{\substack{non-bonded \\ pairs}} \left[\frac{q_i q_j}{4\pi\epsilon_0 r_{ij}} + \frac{A_{ij}}{r_{ij}^{12}} - \frac{B_{ij}}{r_{ij}^{6}}\right] \quad (5)$$

U is the potential energy of the system, k is a force constant r_{ij} and r_0 are the actual and equilibrium bond lengths respectively, $\theta_{ijk} and\ \theta_0$ are bond angle and equilibrium bond angle respectively, $\phi_{ijk\ell}$ is the variation of the torsion angle, δ_n is the torsion equilibrium value. The first term in the equation represents the sum over all bonds with an equilibrium bong-length r_0. The second term is a sum over all bond angles. The third term is the sum over all torsions involving four connected atoms. The fourth term is a sum over the non-bonded interactions (between and within molecules)[17]; this term consists of Coulomb's Law where q_i and q_j are point charges, ϵ_0 is the dialectic constant in vacuum, and r_{ij} is the distance between two point charges. The fourth term also consists of the Lennard-Jones interactions, $\frac{A_{ij}}{r_{ij}^{12}} - \frac{B_{ij}}{r_{ij}^{6}}$ representing the repulsive (Pauli exclusion principle) and attractive interactions (London dispersion forces) between atoms.

2.4 Computational Parameters

2.4.1 Explicit Solvent

For the explicit solvent simulations a protein complex consisting of bovine pancreatic trypsin inhibitor and $\boldsymbol{\beta}$-Trypsin was placed inside a bulk solution of water containing 6707 water molecules. The water box had the following dimensions, length x 59.48 Å, length y 84.38 Å, and length z 62.41 Å. Periodic boundary conditions were used to avoid problems associated with boundary effects. The system was neutralized using sodium. All simulations were performed using Nanoscale Molecular Dynamics (NAMD)[28] Version 2.13 using the CHARMM36 force field. The temperature was initialized at 298K and the simulations were performed under constant pressure and temperature using the Langevin Piston and Langevin thermostat, respectively. Visualization of trajectories were done using Visual Molecular Dynamics (VMD)[29].

2.4.2 Generalized Born Implicit Solvent

For the Implicit solvent simulations the solvent was simulated as a continuum medium using the generalized born implicit method. The generalized born equation is an approximation of the Poisson Boltzmann equation. In the generalized born (GB) simulation, the total electrostatic force on an atom is the net Coulomb force on an atom minus the GB force on an atom.[1,2,15,16]

$$\vec{F_i} = \vec{F_i}_{\text{Coulomb}} - \vec{F_i}_{\text{GB}} \quad (6)$$

The GB force on an atom is the derivative of the total GB energy with respect to atom distances. [1,2,15,16] All simulations were performed using Nanoscale Molecular Dynamics (NAMD)[28] version 2.13 using the CHARMM36 force field. The temperature was initialized at 298K and the simulations were performed under constant temperature using the Langevin thermostat. Visualization of trajectories were done using Visual Molecular Dynamics (VMD)[29].

2.5 Methods of Analysis

To determine an estimation of the binding free energy constraints were applied to the protein complex in the Explicit and Implicit simulations. Bovine pancreatic trypsin inhibitor and $\boldsymbol{\beta}$-Trypsin were forcefully separated at the following distances; 10Å, 15Å and 20Å.

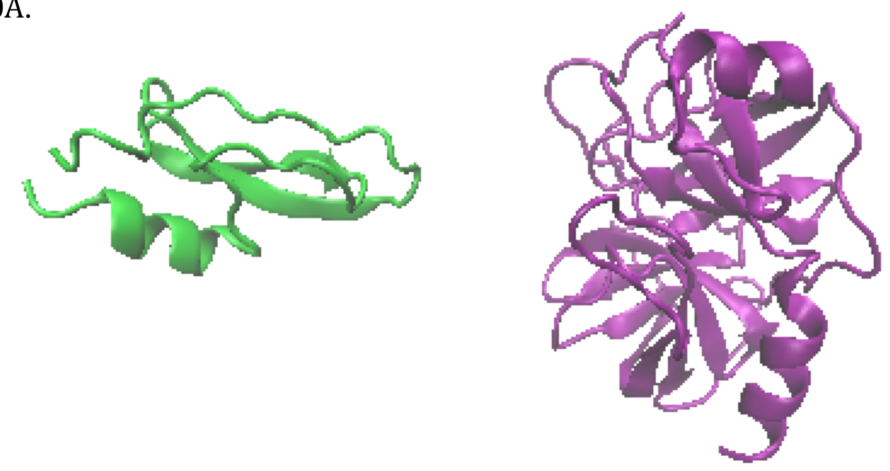

Figure 2: The protein complex, bovine pancreatic trypsin inhibitor (green) and $\boldsymbol{\beta}$-Trypsin (purple) separated by 15Å, structure generated using VMD software.[29]

The potential energies for each run were averaged and the differences between each constraint distance were taken for the implicit and explicit models. These values were compared as a rough estimate of the binding free energies of the protein complex. This was done in place of other methods due to the time constraints of the thesis. These values are limited in their accuracy but offer an idea of the accuracies of each model.

3 Results and Discussion

3.1 Analysis of Side Chains

Visually analyzing the snapshots obtained from the explicit and implicit models it can be see that the solvent has an effect on how the side chains are situated. These differences can be seen in Figure 3 and 4.

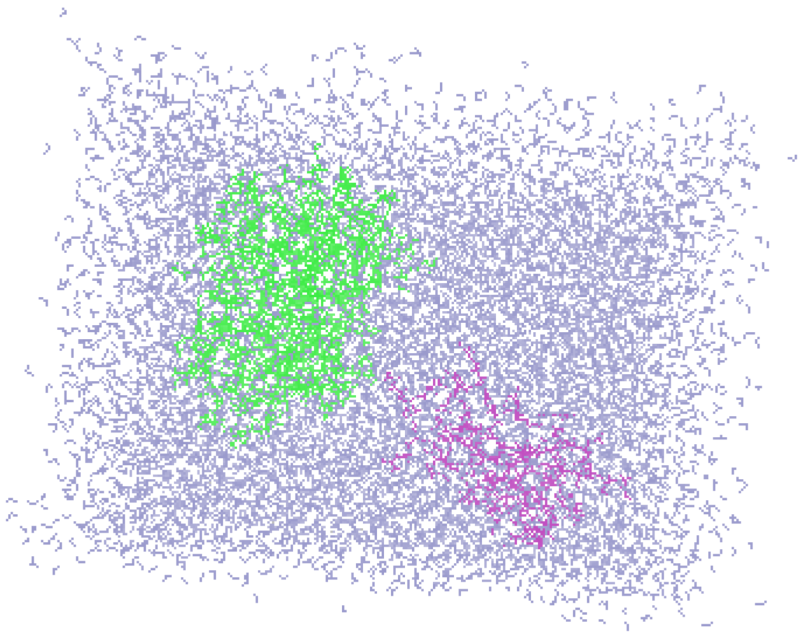

Figure 3: The protein complex, bovine pancreatic trypsin inhibitor (green) and $\boldsymbol{\beta}$-Trypsin (purple) separated by 10Å in explicit solvent. Structure generated using Visual Molecular Dynamics (VMD) software.[29]

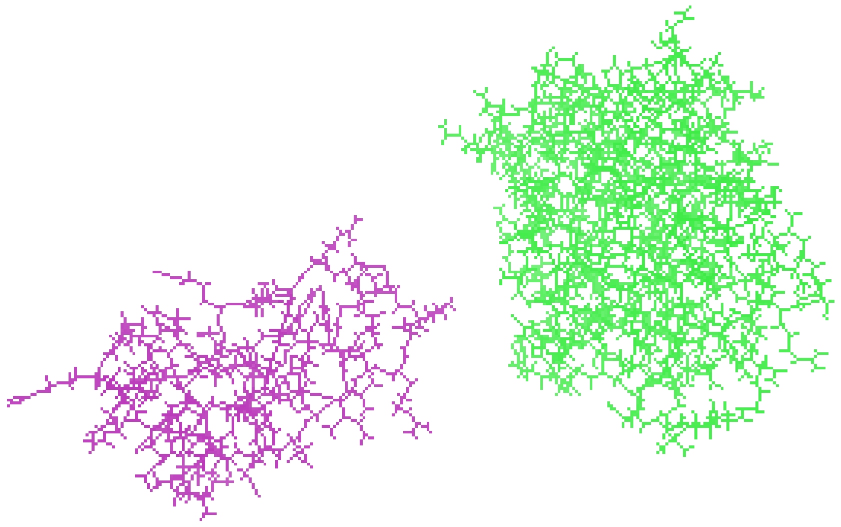

Figure 4: The protein complex, bovine pancreatic trypsin inhibitor (green) and ***β***-Trypsin (purple) separated by 10Å in GB implicit solvent. Structure generated using VMD software.[29]

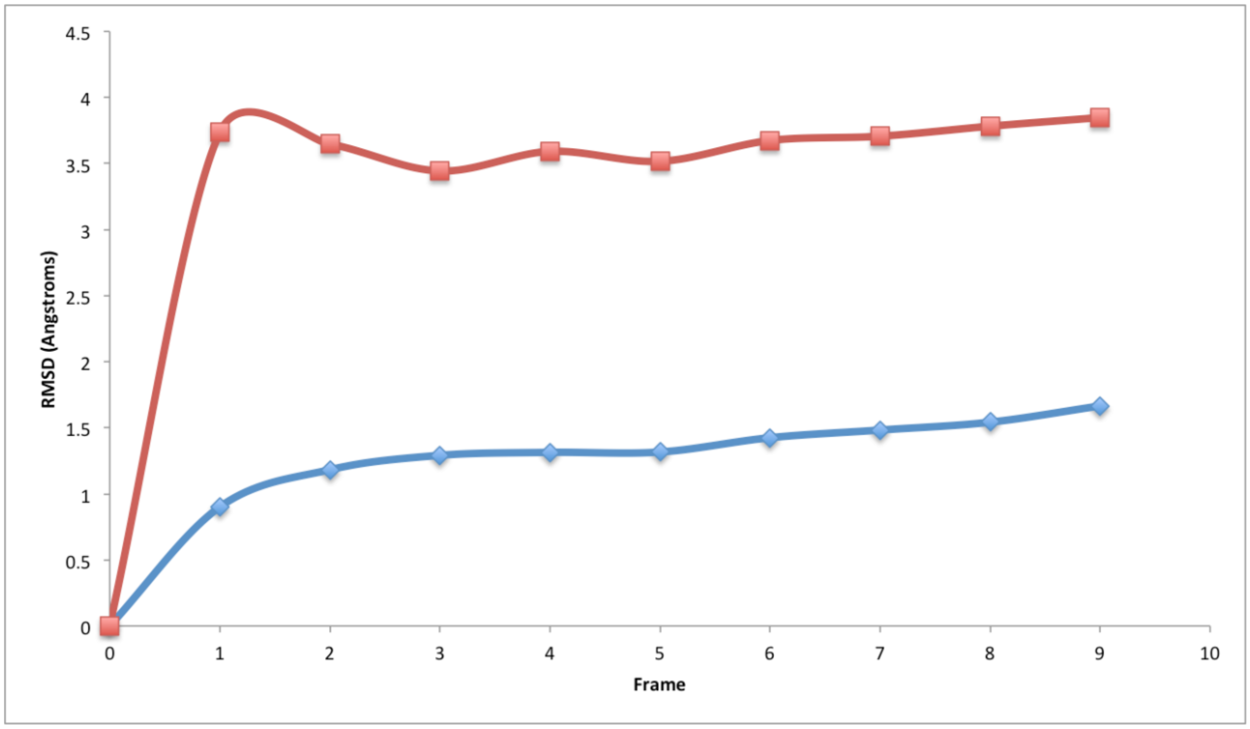

Figure 5: RMSD plot of the protein complex, bovine pancreatic trypsin inhibitor with ***β***-Trypsin in GB implicit solvent (blue) and explicit solvent (red).

Visual inspection of the protein trajectories along with RMSD calculations in explicit and implicit solvent indicates that the modelling of the solvent affects how spread out the side chains are. In the implicit model shown in Figure 3 the structure looks more compact than that of the structure in the explicit model shown in Figure 4. This was further proven through the Root-mean-square deviation of atomic positions (RMSD) plot shown in Figure 5. The protein complex in the explicit solvent had a much larger RMSD then the protein complex in the implicit solvent, therefore it can be concluded that the protein is more compact in the implicit solvent and it has a more open structure in the explicit. These figures demonstrate the effect the solvent has on the shape of the protein complex.

3.2 Computation of the Radial Distribution Function

A Radial distribution function (RDF) was obtained to examine the structural properties of the solvent molecules in the explicit model. The oxygen-oxygen pair interactions between molecules were obtained. RDF's were computed for each of the explicit models at varying distances. Each oxygen-oxygen RDF obtained was identical with no changes, indicating that the constraints added to separate each part of the protein complex did not affect the structural properties of the solvent box, as this can be seen in figure 5.

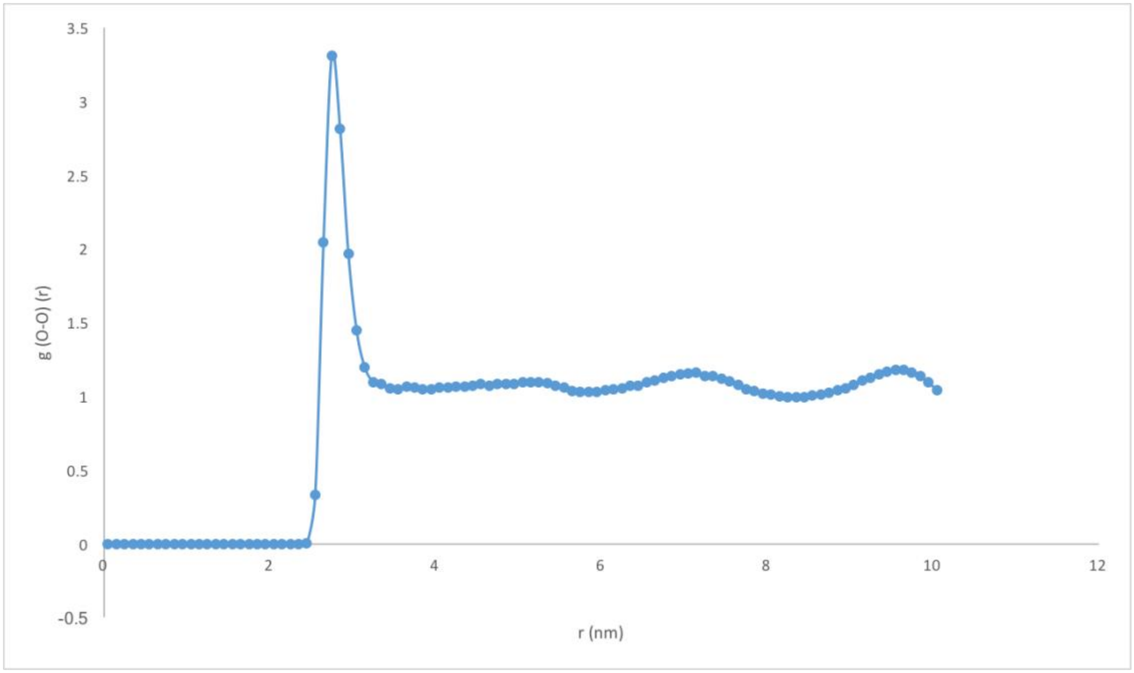

Figure 6: RDF of oxygen-oxygen interaction of solvent water molecules in the presence of bovine pancreatic trypsin inhibitor and $\boldsymbol{\beta}$-Trypsin.

Based on the oxygen-oxygen RDF's obtained, there is an indication that the structure of the water molecules with respect to their location in the simulation box is identical in all of my explicit solvent runs, regardless of constraint distance. The RDF does not provide possible changes in the orientation of the water molecules because of the presence of the protein complex.

3.3 An Estimation of Binding Free Energy

The potential energies were determined for each run at each distance, separated for the explicit and implicit model.

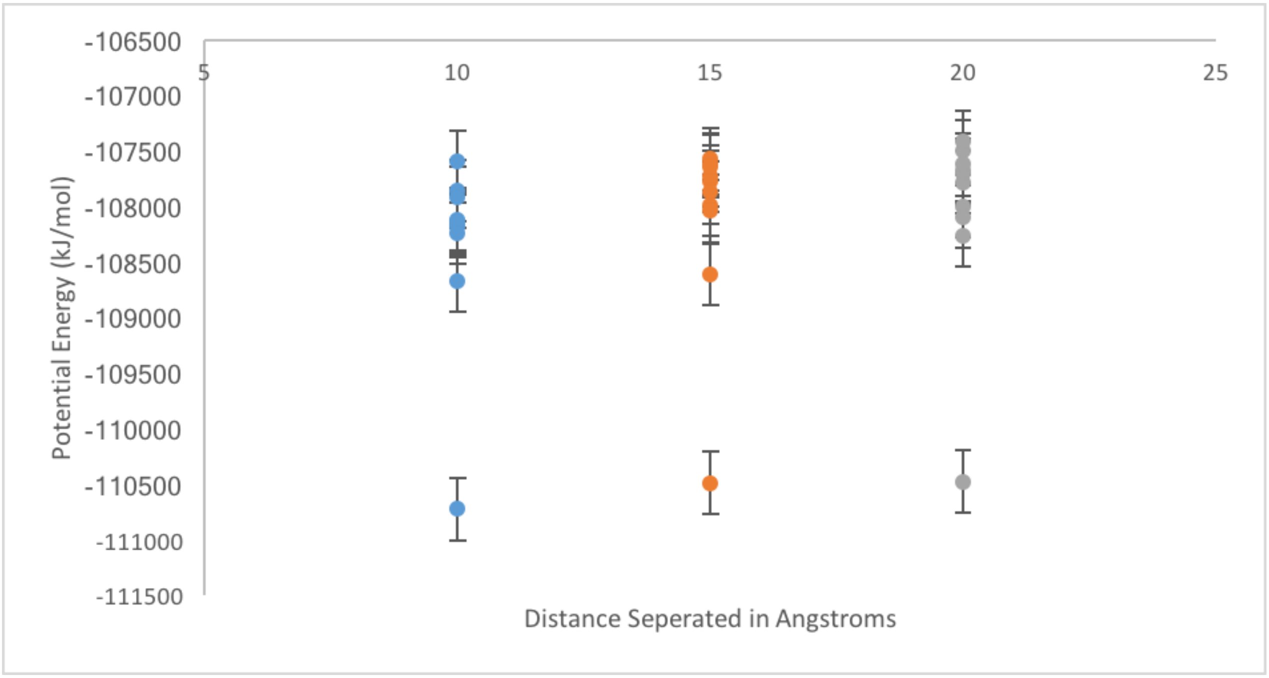

Figure 7: Potential energies obtained from each run at different constrained distances for the explicit model.

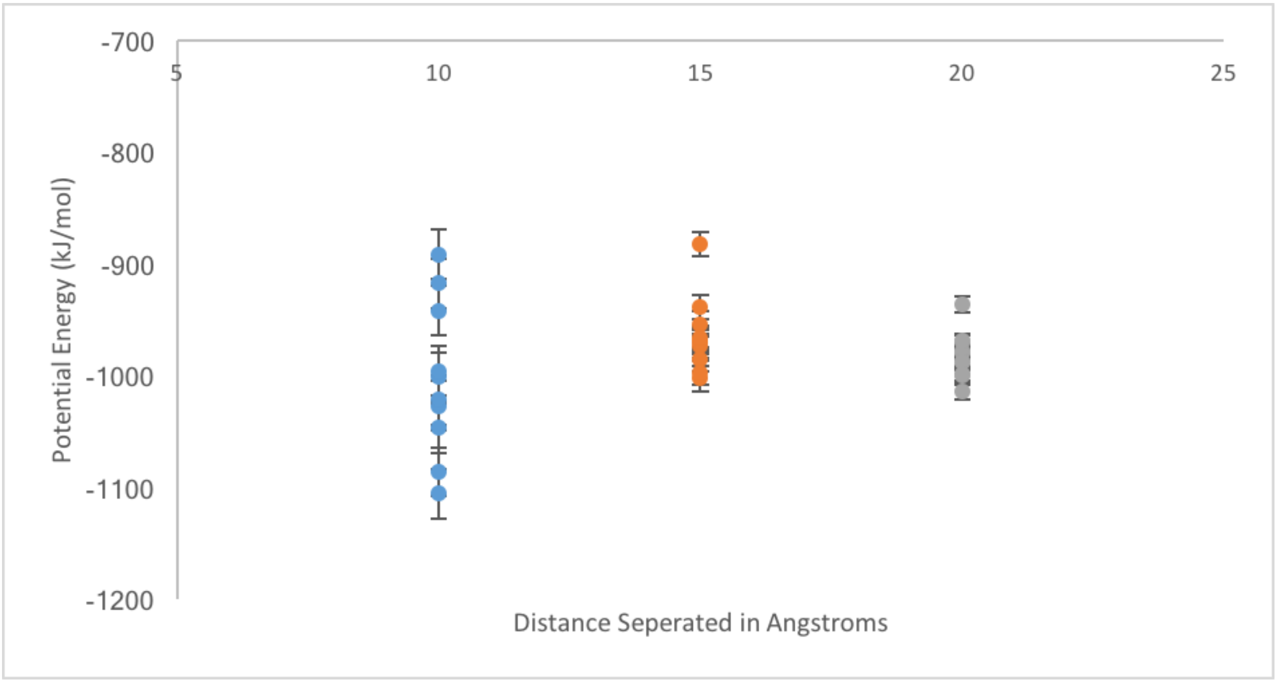

Figure 8: Potential energies obtained from each run at different constrained distances for the implicit model.

From Figures 3 and 4, it is seen that the constraints did not have a large effect on the potential energy of the system. However, there are large differences in the potential energies seen between the explicit and implicit models, which was expected since this was seen in the literature. Next, the potential energies were then averaged at each distance for each solvent model.

Distances separated	10Å	15Å	20Å
Explicit Model Average Potential (kJ/mol)	-108356.0274	-108130.6349	-108049.7169
Implicit model Average Potential (kJ/mol)	-1003.3789	-963.62895	-981.99814

Table 1. The average potentials of each run, with each part of the protein complex being held apart at varying distances.

The averages of the potential energies were used to determine a rough estimation of binding free energy by determining the differences between the average potential energy at each distance for each solvent model. The reference value was at 10 Å.

From reference value at 10Å at:	15Å	20Å
Explicit Model Difference in Average Potential (kcal/mol)	53.9	73.2
Implicit model Difference in Average Potential (kcal/mol)	9.5	5.1

Table 2: The difference in average potential energy in kcal/mol from a distance of 10 Å

Krowarsch et al. experimentally determined the binding free energies of a variety of protein complexes, one of which was the protein complex bovine pancreatic trypsin inhibitor binding with $\boldsymbol{\beta}$-Trypsin[11]. They found that the complex of interest had a binging free energy of 15 kcal/mol[11]. The typical standard that is set for error within computer simulations for determining binding free energy is 1 kcal/mol[7], therefore neither the values determined for the explicit or implicit model agreed with the experimental value. There could be many reasons for this, including the way in which binding free energy was determined for this experiment. Typically other methods are used for determining this energy value, for example the Free energy perturbation (FEP) method, which was deemed one of the most accurate methods to determine binding free energy in the literature[19]. However, due to the time constraints of this fourth year thesis project a different method had to be employed.

4 Conclusion

In this thesis, molecular dynamics simulations were conducted on the protein complex bovine pancreatic trypsin inhibitor binding with $\boldsymbol{\beta}$-Trypsin. These simulations were carried out in an explicit solvent using TIP3P water, and in an implicit solvent based on the generalized born model. It was found that there are clear differences in the potential energies between values obtained from the implicit and explicit simulations, which was to be expected based on the literature review. However, neither model provided agreeable results with the experimental binding free energy value of 15kcal/mol[11]. This was most likely due to the methods used to determine the binding free energy. There were limitations to the method employed, therefore my results are an estimation rather then an accurate depiction of the binding free energy values. Lastly, this estimation still proved to be useful to my analysis because a large difference in values could be seen in the estimated binding free energies between the implicit and explicit model.

In the future, to expand on this work, longer simulations could be run and the Free energy perturbation (FEP) method could be used to calculate binding free energies, as it was the most reliable method seen in the literature[19,22]. Also, simulations could be run, testing different implicit and explicit models using the same protein complex to determine which model provides the most accurate results for this large biomolecule.

References

[1] D. Bashford, D. A. Case, Physcial Chem. **2000**, 51, 129–152.

[2] L. Benedict, C. Christophe, E. Ron, A. Laaksonen, A. Mark, T. Schlick, C. Schütte, R. Skeel, New Algorithms for Macromolecular Simulation, Springer, **2006**.

[3] J. Chen, Y. Shao, J. Ho, J. Phys **2019**, 123, 5580–5589.

[4] A. Cumberworth, J. M. Bui, J. Gsponer, J. Comput. Chem. **2016**, 37, 629–640.

[5] C. J. Fennell, K. A. Dill, **2011**, 209–226.

[6] D. Frenkel, Understanding Molecular SimulationFrom Algorithms to Application, Academic Press, **1996**.

[7] S. Izadi, B. Aguilar, A. V. Onufriev, J. Chem. Theory Comput. **2015**, 11, 4450–4459.

[8] S. Izadi, R. Anandakrishnan, A. V Onufriev, Phys. Chem. **2014**, 1–30.

[9] E. V Katkova, A. V Onufriev, B. Aguilar, V. B. Sulimov, **2018**, 70–80.

[10] J. Kleinjung, W. R. P. Scott, J. R. Allison, W. F. Van Gunsteren, F. Fraternali, J. Chem. Theory Comput. **2012**, 8, 2391–2403.

[11] D. Krowarsch, M. Dadlez, O. Buczek, I. Krokoszynska, A. O. Smalas, J. Otlewski, J. Mol. Biol. **1999**, 289, 175–186.

[12] T. Lazaridis, Proteins Struct. Funct. Genet. **2003**, 52, 176–192.

[13] T. Lazaridis, M. Karplus, **1999**, 152, 133–152.

[14] D. L. Mobley, C. I. Bayly, M. D. Cooper, M. R. Shirts, K. A. Dill, J. Chem. Theory Comput. **2009**, 5, 350–358.

[15] A. Onufriev, **2010**, 1–52.

[16] A. V. Onufriev, D. A. Case, Annu. Rev. Biophys. **2019**, 48, 275–296.

[17] M. P. Allen, D. J . Tildesley, Computer Simulation of Liquids, Oxford University Press, **2017**.

[18] N. V Prabhu, P. Zhu, K. I. M. A. Sharp, **2004**, DOI 10.1002/jcc.20138.

[19] R. J. Radmer, P. A. Kollman, **1996**, 18, 902-919

[20] B. Roux, T. Simonson, Biophys. Chem. **1999**, 78, 1–20.

[21] C. Sagui, T. A. Darden, Annu. Rev. Biophys. Biomol. Struct. **1999**, 28, 155–179.

[22] D. Shivakumar, Y. Deng, B. Roux, J. Chem. Theory Comput. **2009**, 5, 919–930.

[23] B. A. Sorenson, S. S. Hong, H. C. Herbol, P. Clancy, **2019**, 170.

[24] S. Steinmann, P. Sautet, C. Michel, **2016**, 18, 31850–31861.

[25] C. Tan, L. Yang, R. Luo, **2006**, 18680–18687.

[26] J. Zhang, H. Zhang, T. Wu, Q. Wang, D. Van Der Spoel, **2017**, DOI 10.1021/acs.jctc.7b00169.

[27] L. Y. Zhang, E. Gallicchio, R. A. Friesner, R. M. Levy, J. Comput. Chem. **2001**, 22, 591–607.

[28] J. C. Phillips, R. Braun, W. Wang, J. Gumbart, E. Tajkhorshid, E. Villa, C. Chipot, R. D. Skeel, L. Kale, and K. Schulten. *Journal of Computational Chemistry.* **2005**, 26:1781-1802.

[29] Humphrey, W., Dalke, A. and Schulten, K., ``VMD - Visual Molecular Dynamics" *J.Molec. Graphics* **1996,** *14.1*, 33-38.

Appendix 1

Molecular Dynamics Code written in C++

```
//
//  Molecular Simulation.hpp
//  Header File

#ifndef Molecular_Simulation_hpp
#define Molecular_Simulation_hpp

#include <stdio.h>
#include <iostream>
#include <cmath>
#include <cstdlib>
#include <ctime>
#include <fstream>
#include <string>

double instT;

typedef struct
{
   const double epsilon = 1.0; //J
   const double sigma = 0.341; //Van der Waals radius Ar 0.341nm or 3.41
angstroms
   const double dt = 0.001;
   const double dlatt = 1.0;
}simConst;
```

```
typedef struct
{
    int x;
    int y;
    int z;
} cubelength;

typedef struct
{
    int argonMolecules;
    int timeSteps;
    double temperature;
    double tReal;
    cubelength cube;
}inputValues;

typedef struct
{
    double kinetic = 0.0;
    double potential = 0.0;
    double total = 0.0;
} energy;

typedef struct // x - Poitions, y - Velocities, z - Forces
{
    double x;
    double y;
    double z;
    double prevX;
    double prevY;
    double prevZ;
    double futX;
    double futY;
    double futZ;
} molecularMatrix;

typedef struct
{
    molecularMatrix posR;
    molecularMatrix velV;
    molecularMatrix forF;
} atom;

typedef struct
{
    //file output Names;
    std::string trajFout = "trajFout.txt";
}IOFiles;

typedef struct
{
    double rtmpX = 0.0;
    double rtmpY = 0.0;
    double rtmpZ = 0.0;
} intitialPosition;

typedef struct
{
    double rIJ;
    double forceIJ;
    double dfX;
    double dfY;
    double dfZ;
    double forceX;
    double forceY;
    double forceZ;
    double r = 0;
    double rS;
    double sigmaS;
    double sigmaVal;
} initialForce;

typedef struct
{
    double r2;
    double vx;
    double vy;
    double vz;
    double dx;
    double dy;
    double dz;
} initialEnergy;

typedef struct
{
    double sumVelSqd = 0.0;
    double tFactor;
} temperatureRescale;


void iPosition();
void iForce();
void iEngery();
void Verlet();
void tRescale();

#endif /* Molecular_Simulation_hpp */








//
//  Molecular Simulation.cpp
//  Source Code
//
```

```
#include "Molecular Simulation.hpp"

using namespace std;

//Global Variables and Constants
inputValues userInput;
simConst fixedVal;
energy setEnergyValues;
intitialPosition initPos;
initialForce initFor;
initialEnergy initEne;
temperatureRescale tempRescale;
IOFiles readWriteTo;

atom aMolecularValues[50000];

ofstream fileOut(readWriteTo.trajFout);

// Compute Cube Size and Atom Initial Position
void iPosition()
{
   int i, j, k, l = 0;

   for (i = 0; i < userInput.cube.x; i++)
   {
      initPos.rtmpX = i * fixedVal.dlatt;
```

```
        for (j = 0; j < userInput.cube.y; j++)
        {
            initPos.rtmpY = j * fixedVal.dlatt;
            for (k = 0; k < userInput.cube.z; k++)
            {
                initPos.rtmpZ = k * fixedVal.dlatt;
            }
        }
    }

    for (i = 0; i < userInput.argonMolecules; i++)
    {
        aMolecularValues[l].posR.x = initPos.rtmpX;
        aMolecularValues[l].posR.y = initPos.rtmpY;
        aMolecularValues[l].posR.z = initPos.rtmpZ;
        l++;

        aMolecularValues[i].posR.prevX = aMolecularValues[i].posR.x;
        aMolecularValues[i].posR.prevY = aMolecularValues[i].posR.y;
        aMolecularValues[i].posR.prevZ = aMolecularValues[i].posR.z;
        aMolecularValues[i].velV.x = 0.0;
        aMolecularValues[i].velV.y = 0.0;
        aMolecularValues[i].velV.z = 0.0;

        cout << "Positions" << "\t" << aMolecularValues[i].posR.x << "\t" <<
aMolecularValues[i].posR.y << "\t" << aMolecularValues[i].posR.z << endl;
```

```
        cout << "Velocities" << "\t" << aMolecularValues[i].velV.x << "\t" <<
aMolecularValues[i].velV.y << "\t" << aMolecularValues[i].velV.z << endl;
    }
}

// Verlet Algorithm
void Verlet()
{
    int i;
    int iSteps;
    for (iSteps = 0; iSteps < userInput.timeSteps; iSteps++)
    {
        iForce();
        fileOut << userInput.argonMolecules << "\n" << endl;

        for (i = 0; i < userInput.argonMolecules; i++)
        {
            aMolecularValues[i].posR.futX = 2 * aMolecularValues[i].posR.x -
aMolecularValues[i].posR.prevX + (aMolecularValues[i].forF.x) * fixedVal.dt *
fixedVal.dt;
            aMolecularValues[i].posR.futY = 2 * aMolecularValues[i].posR.y -
aMolecularValues[i].posR.prevY + (aMolecularValues[i].forF.y) * fixedVal.dt *
fixedVal.dt;
            aMolecularValues[i].posR.futZ = 2 * aMolecularValues[i].posR.z -
aMolecularValues[i].posR.prevZ + (aMolecularValues[i].forF.z) * fixedVal.dt *
fixedVal.dt;
```

```
            aMolecularValues[i].velV.x = 0.5 * (aMolecularValues[i].posR.futX -
aMolecularValues[i].posR.prevX) / fixedVal.dt;
            aMolecularValues[i].velV.y = 0.5 * (aMolecularValues[i].posR.futY -
aMolecularValues[i].posR.prevY) / fixedVal.dt;
            aMolecularValues[i].velV.z = 0.5 * (aMolecularValues[i].posR.futZ -
aMolecularValues[i].posR.prevZ) / fixedVal.dt;

            fileOut << "Ar \t\t" << aMolecularValues[i].posR.x << "\t"
<<aMolecularValues[i].posR.y <<"\t" << aMolecularValues[i].posR.z << endl;

            aMolecularValues[i].posR.prevX = aMolecularValues[i].posR.x;
            aMolecularValues[i].posR.prevY = aMolecularValues[i].posR.y;
            aMolecularValues[i].posR.prevZ = aMolecularValues[i].posR.z;
            aMolecularValues[i].posR.x = aMolecularValues[i].posR.futX;
            aMolecularValues[i].posR.y = aMolecularValues[i].posR.futY;
            aMolecularValues[i].posR.z = aMolecularValues[i].posR.futZ;
        }

        iEngery();
    }
    cout << "Verlet is finished";
}

// Compute Force
void iForce()
{
    int i, j = 0;
    for (i = 0; i < userInput.argonMolecules; i++)
    {
        aMolecularValues[i].forF.x = 0.0;
        aMolecularValues[i].forF.y = 0.0;
        aMolecularValues[i].forF.z = 0.0;
    }

    for (i = 0; i < userInput.argonMolecules - 1; i++) // Loop the atoms
    {
        for (j = i + 1; j < userInput.argonMolecules; j++)
        {


            initFor.r = initFor.r + 0.1;
            initFor.sigmaS = fixedVal.sigma * fixedVal.sigma;
            initFor.rS = initFor.r * initFor.r;
            initFor.sigmaVal = initFor.sigmaS / initFor.rS;

            initFor.forceIJ  = ((-24 * fixedVal.epsilon) / (initFor.rS)) * ((-2 *
(pow(initFor.sigmaVal,6)) + pow(initFor.sigmaVal,3)));

            aMolecularValues[j].posR.x = aMolecularValues[i].posR.x * initFor.forceIJ;
            aMolecularValues[j].posR.y = aMolecularValues[i].posR.y * initFor.forceIJ;
            aMolecularValues[j].posR.z = aMolecularValues[i].posR.z * initFor.forceIJ;

            initFor.dfX = (aMolecularValues[i].posR.x - aMolecularValues[j].posR.x);
            initFor.dfY = (aMolecularValues[i].posR.y - aMolecularValues[j].posR.y);
            initFor.dfZ = (aMolecularValues[i].posR.z - aMolecularValues[j].posR.z);

            initFor.forceX = initFor.forceIJ * initFor.dfX;
            initFor.forceY = initFor.forceIJ * initFor.dfY;
            initFor.forceZ = initFor.forceIJ * initFor.dfZ;

            aMolecularValues[i].forF.x = aMolecularValues[i].forF.x + initFor.forceX;
            aMolecularValues[i].forF.y = aMolecularValues[i].forF.y + initFor.forceY;
            aMolecularValues[i].forF.z = aMolecularValues[i].forF.z + initFor.forceZ;
            aMolecularValues[j].forF.x = aMolecularValues[j].forF.x - initFor.forceX;
            aMolecularValues[j].forF.y = aMolecularValues[j].forF.y - initFor.forceY;
            aMolecularValues[j].forF.z = aMolecularValues[j].forF.z - initFor.forceZ;
        }
    }

    for (i = 0; i < userInput.argonMolecules; i++)
    {
        cout << "iForce" << endl;
        cout << aMolecularValues[i].forF.x << "\t" << aMolecularValues[i].forF.y <<
"\t" << aMolecularValues[i].forF.z << endl;

    }
}

// Compute Leonard Jones Potential for atom interactions
void iEngery()
{
   int i;
   int j;

   for (i = 0; i < userInput.argonMolecules; i++)
   {
      initEne.vx = aMolecularValues[i].velV.x;
      initEne.vy = aMolecularValues[i].velV.y;
      initEne.vz = aMolecularValues[i].velV.z;
      setEnergyValues.kinetic += 0.5 * (initEne.vx * initEne.vx + initEne.vy *
initEne.vy + initEne.vz * initEne.vz);
   }

   for (i = 0; i < userInput.argonMolecules - 1; i++)
   {
      for (j = i + 1; j < userInput.argonMolecules; j++)
      {
         initEne.dx = aMolecularValues[i].posR.prevX -
aMolecularValues[j].posR.prevX;
         initEne.dy = aMolecularValues[i].posR.prevY -
aMolecularValues[j].posR.prevY;
         initEne.dz = aMolecularValues[i].posR.prevZ -
aMolecularValues[j].posR.prevZ;
         initEne.r2 = initEne.dx * initEne.dx + initEne.dy * initEne.dy + initEne.dz *
initEne.dz;

         setEnergyValues.potential += 4.0 * fixedVal.epsilon * (pow(initEne.r2, -
6.0) - pow(initEne.r2, -3.0));
```

```
        }
    }

    instT = ((2.0 * setEnergyValues.kinetic) / (3.0 * userInput.argonMolecules));
    setEnergyValues.total = setEnergyValues.kinetic + setEnergyValues.potential;

    cout << "Total Energy" << "\t" << setEnergyValues.kinetic << "\t" << instT
<<"\t" << setEnergyValues.potential << "\t" << endl;
    cout << setEnergyValues.total << endl;
}

void tRescale()
{
    int i;

    for (i = 0; i < userInput.argonMolecules; i++)
    {
        tempRescale.sumVelSqd += (aMolecularValues[i].velV.x *
aMolecularValues[i].velV.x) +(aMolecularValues[i].velV.y *
aMolecularValues[i].velV.y) + (aMolecularValues[i].velV.z *
aMolecularValues[i].velV.z);
    }

    instT = (tempRescale.sumVelSqd /3 * userInput.argonMolecules);
    tempRescale.tFactor = sqrt(userInput.temperature/instT); // Tfactor is the
scaling factor, which is lambda = sqrt (desired T/inst_T)
```

```
    for (i = 0; i < userInput.argonMolecules; i++)
    {

        aMolecularValues[i].velV.x *= tempRescale.tFactor ;
        aMolecularValues[i].velV.y *= tempRescale.tFactor ;
        aMolecularValues[i].velV.z *= tempRescale.tFactor ;
    }
}

void userIn()
{
    cout << "Enter the number of Argon molecules: \n";
    cin >> userInput.argonMolecules;

    cout << "Enter the number of timesteps: \n";
    cin >> userInput.timeSteps;

    cout << "Enter the the length of each timestep in fs: ";
    cin >> userInput.tReal;

    cout << "Enter size of the starting cubic system: \n";
    cin >> userInput.cube.x;
    cin >> userInput.cube.y;
    cin >> userInput.cube.z;

    cout << "Enter temperature: \n";
    cin >> userInput.temperature;
```

```
}

int main(int argc, char *argv[])
{
    // Execute Functions
    userIn();
    iPosition();
    tRescale();
    Verlet();

    fileOut << "Total Energy" << "\t" << setEnergyValues.kinetic << "\t" << instT <<
"\t" << setEnergyValues.potential << "\t" << endl;
    fileOut << setEnergyValues.total << endl;
    fileOut.close();

    return 0;
}
```