BIRZEIT UNIVERSITY
FACULTY OF GRADUATE STUDIES

COMPARISON OF RESIDUE MOTION CORRELATION IN *BPTI* USING VACUUM AND SOLVENT MOLECULAR DYNAMICS SIMULATIONS

by

Basem Al-'Ajarmeh

A THESIS SUBMITTED
IN PARTIAL FULFILLMENT OF THE
REQUIREMENTS FOR THE DEGREE

MASTER OF SCIENTIFIC COMPUTING

APPROVED, THESIS
COMMITTEE:

Dr. Wael Karain

Dr.Mazen Hamed

Dr.Hani Awad

Birzeit, Palestine

February, 2005

ABSTRACT

Correlated motions in protein molecules play a very important role in the function of the protein. The method of essential dynamics was used in this work to study these correlated motions in the protein Bovine Pancreatic Trypsin Inhibitor(BPTI). Molecular dynamics trajectories were simulated using the program $NAMD$ both in solution and in vacuum for 20ns. Several diagnostic tests were performed on the trajectories: $RMSD$, temperature factor, correlation maps, and essential dynamics(PCA) analysis as well. The results were compared for the two environments, and inspected for their behavior versus time. The main contribution of this work is that this is the longest reported simulation for this protein and offers new insights into how the atoms of the protein behave versus time for "long" simulations.

Acknowledgments

I am very thankful to Dr. Wael Karain for all his help and support as my thesis advisor. I would also like to thank Dr.Hassan Shebli, Dr.Aziz Shawabkeh, Dr. Ali Jaber and the computer department staff. Finally a special thanks to my family.

Contents

Abstract ii

Acknowledgments iii

List of Figures viii

List of Tables xvi

1 Background in Molecular Dynamics 1

1.1 State of the Art in Protein Essential Dynamics 1

1.2 Introduction 1

1.3 Historical Background 4

1.4 Statistical Mechanics 5

1.5 Classical mechanics 6

1.6 Integration Algorithms 9

1.6.1 Verlet 10

1.6.2 The leap-Forg 11

1.6.3 Velocity Verlet 12

1.6.4 Beeman's 12

1.7 Molecular Mechanics and Modeling 13

1.7.1 Introduction 13

1.7.2 Classical force field component 15

1.7.3 The Anatomy of a Molecular Mechanics force field . 16

1.8 Essential Dynamics 28

1.8.1 Background 28

1.8.2 Introduction to ED 29

1.8.3 Multivariate covariance and correlation 30

1.8.4 Principal component analysis PCA 32

1.8.5 Essential Dynamics Technique for Protein 36

1.9 Molecular and Essential Dynamics Tools used 47

1.9.1 $NAMD$: MOLECULAR DYNAMICS SIMULATION 47

1.9.2 VMD:Visual Molecular Dynamics: Computer Tool 49

1.9.3 Unix Cat Trajectory Binary File (Catdcd) 50

1.9.4 Byte Order Reverser Flipdcd 51

1.9.5 matlab and its interrupter Octave 52

1.9.6 Principal Component Analysis Tool (CARMA) . . 53

2 Results and Analysis 55

2.1 Molecular and Essential Dynamics Method used 55

2.1.1 Preparation of the Protein Molecule 56

2.1.2 neutralized of $BPTI$ 61

2.1.3 neutralization of $BPTI$ in a water box 63

2.1.4 Simulation of BPTI in both Vacuum and Solution . 65

2.1.5 Minimization 68

2.1.6 Heat 68

2.1.7 Equilibration 68

2.1.8 Dynamics: Control production phase 69

2.1.9 Analysis 69

2.2 Analysis of results 77

2.2.1 B-Factors 77

2.2.2 Root Mean Square Deviation RMSD: 83

2.2.3 Kinetic Energy: 85

2.2.4 Correlation Maps: 87

2.2.5 Eigenvalues 94

2.2.6 Motions Along Eigenvectors for 20 ns 98

2.2.7 Motions Along Eigenvectors for each ns separately . 105

2.2.8 Trajectory projected on planes 108

2.2.9 Probability distributions of the Motions 117

2.2.10 Projecting several separate eigenvectors onto first ns 123

3 Conclusions 125

A Routine java Code 127

A.1 rotation class 127

A.2 Corrolation and Analysis class 136

A.3 projection class 142

A.4 Extract cα from Trajectory DCD 144

A.5 Convert DCD File Format To PDB format class 145

A.6 Extract Important Data From NAMD Logfile class 146

B Files Format **149**

B.1 *PDB* Format 149

B.2 *PSF* Format 150

B.3 A force field parameter Format 150

B.4 A configuration file Format 151

B.5 Parameter file Format 152

B.6 Topology file Format 152

B.7 DCD trajectory format 152

C *NAMD* Configration File **153**

C.1 *NAMD* Configration File in Vacuum 153

C.2 *NAMD* Configration File in water 154

C.3 PBC Periodic Boundary Condition 161

References **163**

List of Figures

1.1 Molecular dynamics vs Experimental technique; why we need to study bimolecular structure, dynamics, and function 3

1.2 Force field type relation. The mathematical form of the potential energy terms differs from force field to another . 18

1.3 The stretching energy 19

1.4 The stretching two atoms motion 20

1.5 bending energy curve 21

1.6 bending energy angle 21

1.7 Dihedral Energy 22

1.8 Dihedral angle 23

1.9 The electrostatic potential 24

1.10 The electrostatic atom partial charge 25

1.11 Van der Waals interaction energy 26

1.12 Urey-Bradley Energy Potential 26

1.13 improper dihedral energy 27

1.14 Divide the structure overall matrix M into substructure matrices M1, M2, etc, of smaller size that can fit in computer memory 43

1.15 For each substructure matrix M1, M2, etc, find the sum and number of rows in each one 44

1.16 Find the average for the all structure column as X1, X2, etc, and divide it on total number of rows 45

1.17 Find the deviation of each column's structure 46

2.1 Molecular Dynamics technique, MD simulation phases . . . 57

2.2 BPTI in vacuum 66

2.3 BPTI in Solution after ionization 67

2.4 Comparison of rms fluctuations from simulation and from crystallographic B-factors. Fluctuations computed for $C\alpha$ (57 atom) atoms sampled from 0-20 ns water simulation. . 80

2.5 Comparison of rms fluctuations from simulation and from crystallographic B-factors. Fluctuations computed for $C\alpha$ atoms (57 atom) sampled from 0-20 ns vacuum simulation. 81

2.6 The Hunenberger [40] crystallographic B-factors for different window length. 82

2.7 Calculated rms $C\alpha$ deviation (RMSD) of dynamic structures from the initial structure vs. time from the MD simulation for 20ns on $BPTI$. 84

2.8 kinetic energy for 20ns simulation. 86

2.9 Calculated residue-residue-based correlated motions (Dynamic Cross Correlation Maps DCCM) after 1ns simulation for water (Top) and vacuum (Bottom) 88

2.10 Calculated residue-residue-based correlated motions (Dynamic Cross Correlation Maps DCCM) after 5ns simulation for water (Top) and vacuum (Bottom) 89

2.11 Calculated residue-residue-based correlated motions (Dynamic Cross Correlation Maps DCCM) after 6ns simulation for water (Top) and vacuum (Bottom) 90

2.12 Calculated residue-residue-based correlated motions (Dynamic Cross Correlation Maps DCCM) after 10ns simulation for water (Top) and vacuum (Bottom) 91

2.13 Calculated residue-residue-based correlated motions (Dynamic Cross Correlation Maps DCCM) after 15ns simulation for water (Top) and vacuum (Bottom) 92

2.14 Calculated residue-residue-based correlated motions (Dynamic Cross Correlation Maps DCCM) after 20ns simulation for water (Top) and vacuum (Bottom) 93

2.15 Eigenvalues, in decreasing order of magnitude for water (Top) and vacuum (Bottom) simulations obtained from $C\alpha$ coordinates covariance matrix after 1ns. 95

2.16 Eigenvalues, in decreasing order of magnitude for water (Top) and vacuum (Bottom) simulations obtained from $C\alpha$ coordinates covariance matrix after 10ns. 96

2.17 Eigenvalues, in decreasing order of magnitude for water (Top) and vacuum (Bottom) simulations obtained from $C\alpha$ coordinates covariance matrix after 20ns. 97

2.18 Motions of 20ns along first eigenvector obtained from the $C\alpha$ coordinates covariance matrix of the water simulation (Top) and the vacuum simulation (Bottom) 100

2.19 Motions of 20ns along second eigenvector obtained from the $C\alpha$ coordinates covariance matrix of the water simulation (Top) and the vacuum simulation (Bottom) 101

2.20 Motions of 20ns along third eigenvector obtained from the $C\alpha$ coordinates covariance matrix of the water simulation (Top) and the vacuum simulation (Bottom) 102

2.21 Motions of 20ns along twentieth eigenvector obtained from the $C\alpha$ coordinates covariance matrix of the water simulation (Top) and the vacuum simulation (Bottom) 103

2.22 Motions of 20ns along fiftieth eigenvector obtained from the $C\alpha$ coordinates covariance matrix of the water simulation (Top) and the vacuum simulation (Bottom) 104

2.23 Motions along several eigenvectors (1,2,3,50) obtained from the $C\alpha$ 1ns coordinates covariance matrix of the water simulation (upper row) and the vacuum simulation (lower row), (A) 1ns motions after the first ns, (B) 1ns motions after the fifth ns (C) 1ns motions after the tenth ns (D) 1ns motions after the fifteenth ns (E) 1ns motions after the twentieth ns 107

2.24 Projection of the trajectory (water simulation in Top and vacuum simulation Bottom) on the planes defined by two eigenvectors from the $C\alpha$ coordinates covariance matrix. (A) Horizontal axis: displacement along first eigenvector. Vertical axis: displacement along second eigenvector. . . . 110

2.25 Projection of the trajectory (water simulation in Top and vacuum simulation Bottom) on the planes defined by two eigenvectors from the $C\alpha$ coordinates covariance matrix. (A) Horizontal axis: displacement along first eigenvector. Vertical axis: displacement along third eigenvector. 111

2.26 Projection of the trajectory (water simulation in Top and vacuum simulation Bottom) on the planes defined by two eigenvectors from the $C\alpha$ coordinates covariance matrix.(A) Horizontal axis: displacement along second eigenvector. Vertical axis: displacement along third eigenvector. 112

2.27 Projection of the trajectory (water simulation in Top and vacuum simulation Bottom) on the planes defined by two eigenvectors from the $C\alpha$ coordinates covariance matrix.(A) Horizontal axis: displacement along first eigenvector. Vertical axis: displacement along fourth eigenvector. 113

2.28 Projection of the trajectory (water simulation in Top and vacuum simulation Bottom) on the planes defined by two eigenvectors from the $C\alpha$ coordinates covariance matrix.(A) Horizontal axis: displacement along first eigenvector. Vertical axis: displacement along fifth eigenvector. 114

2.29 Projection of the trajectory (water simulation in Top and vacuum simulation Bottom) on the planes defined by two eigenvectors from the $C\alpha$ coordinates covariance matrix.(A) Horizontal axis: displacement along first eigenvector. Vertical axis: displacement along twentieth eigenvector. 115

2.30 Projection of the trajectory (water simulation in Top and vacuum simulation Bottom) on the planes defined by two eigenvectors from the $C\alpha$ coordinates covariance matrix. (A) Horizontal axis: displacement along twentieth eigenvector. Vertical axis: displacement along fiftieth eigenvector. 116

2.31 Probability distributions for the displacements along first eigenvectors for both the water simulation (Top) and the vacuum simulation (Bottom). 118

2.32 Probability distributions for the displacements along second eigenvectors for both the water simulation (Top) and the vacuum simulation (Bottom). 119

2.33 Probability distributions for the displacements along third eigenvectors for both the water simulation (Top) and the vacuum simulation (Bottom). 120

2.34 Probability distributions for the displacements along twentieth eigenvectors for both the water simulation (Top) and the vacuum simulation (Bottom). 121

2.35 Probability distributions for the displacements along fiftieth eigenvectors for both the water simulation (Top) and the vacuum simulation (Bottom). 122

2.36 Projection of one set of eigenvectors onto another for both water (left column) and vacuum simulation (right column). (A) Eigenvectors calculated from first ns on second ns. (B) Eigenvectors calculated from first ns on third ns. (C) Eigenvectors calculated from first ns on fifth ns. (D) Eigenvectors calculated from first ns on tenth ns. (E) Eigenvectors calculated from first ns on fifteenth ns. (F) Eigenvectors calculated from first ns on twentieth ns. 124

B.1 The Protein Data Bank 149

C.1 In Periodic boundary condition when a particle moves in the central box, its periodic image in every one of the other boxes moves with exactly the same orientation in exactly the same way. 162

List of Tables

Chapter 1
Background in Molecular Dynamics

1.1 State of the Art in Protein Essential Dynamics

Essential dynamics (PCA) is an essential tool for studying motions of proteins[1, 2, 3, 4, 5, 6, 7]. Saarela *et, al.* [1] employed (PCA) to show that motions of native BPTI are more correlated than those of its mutants in 1 ns (nanosecond) simulations . Arcangeli *et. al.*[8] also used it to study correlated motions in copper protein plastocyanin and azurin[9] using 1.1 ns time scale (internal) simulation.

This study presents what we believe is the longest reported essential dynamics study for BPTI. It should add new insight information to the intense research field of protein dynamics

1.2 Introduction

Understanding how a protein functions require knowledge of the internal motions of its atoms [10]. Recent evidence indicates that the catalytic activity of an enzyme depends on such correlated motions [11]. These motions range from hundredths ($10^{-2} A^{o}$) to tens of an angstrom in magnitude, and from less than picosecond to several seconds in duration [12].

Progress in fighting diseases or building antibodies that have the ability to deal with and minimize a virus's activity depends on the understanding of such motions.

Experimental techniques such as X-Ray diffraction, NMR, Neutron Scattering are used to study the biomolecular structure, dynamics, and function (Fig 1.1). Theoretical methods are used in parallel with these experimental techniques. One of the principal tools now used in the theoretical study of biological molecules is molecular dynamics (MD) simulations. This computational method calculates the time dependent activities and behavior of a molecular system. MD simulations offer complete information on the fluctuations and conformational changes (Essential Dynamics) of proteins and nucleic acids [13].

Molecular dynamics simulates the motions of a system of particles. It begins with the energy of the system as a function of atomic coordinates. The potential energy function is approximate, but it can be easily modified or tinkered with in order to see how a certain parameter affects the results. The equations of motion are used to calculate the position of each atom as a function of time [12].

Simulation of a protein system at the atomic level could retrieve quantitative and / or qualitative information about the structure function relationship of the protein. Newtonian Dynamics are used in molecular

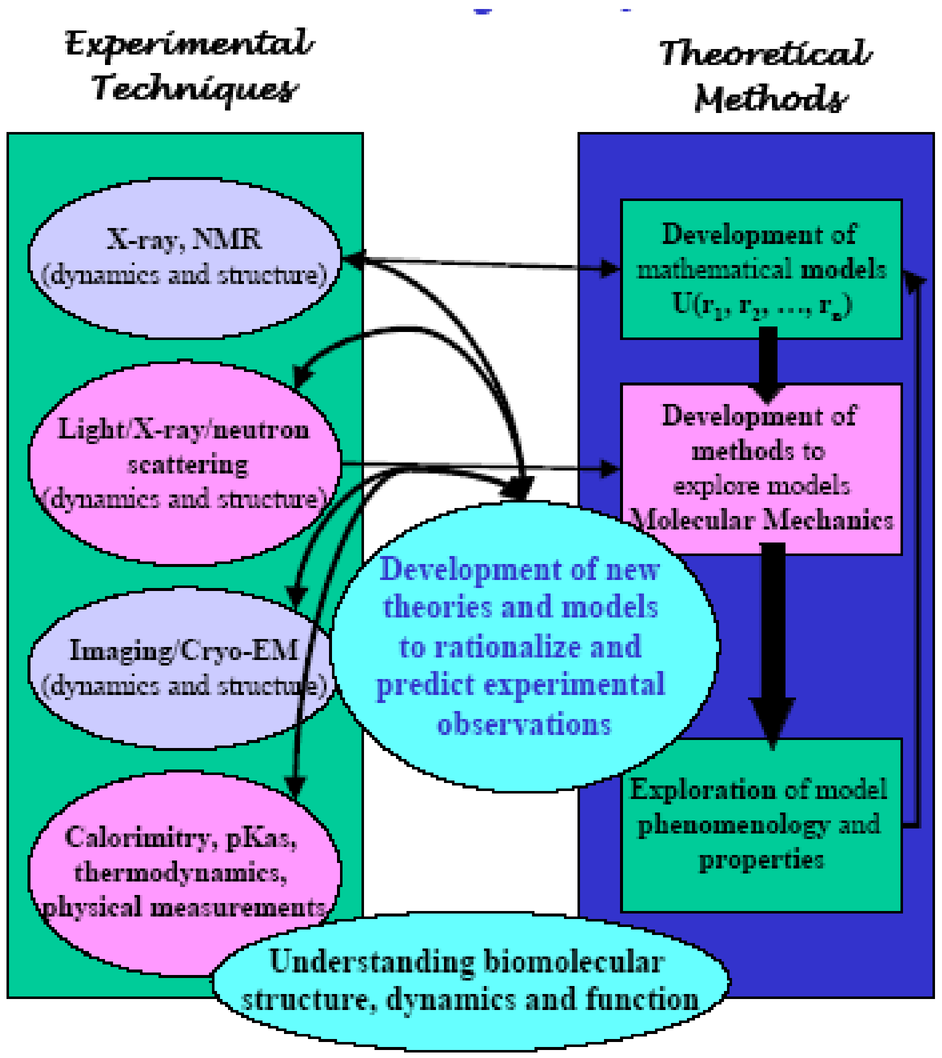

Experimental Techniques
Theoretical Methods
X-ray, NMR (dynamics and structure)
Light/X-ray/neutron scattering (dynamics and structure)
Imaging/Cryo-EM (dynamics and structure)
Calorimitry, pKas, thermodynamics, physical measurements
Development of mathematical models $U(r_1, r_2, \ldots, r_n)$
Development of methods to explore models Molecular Mechanics
Exploration of model phenomenology and properties
Development of new theories and models to rationalize and predict experimental observations
Understanding biomolecular structure, dynamics and function

Figure 1.1 Molecular dynamics vs Experimental technique; why we need to study bimolecular structure, dynamics, and function

dynamics simulation to calculate the motion of atoms by determining the net force and acceleration experienced by each atom.

1.3 Historical Background

The concept of Molecular Dynamics (MD) simulations was introduced in 1957 to study-hard sphere interactions[14]. This led to many important approaches relating to the behavior of simple liquids [15]. A study of the creation of defects induced by radiation damage using MD was performed in 1960. This study was one of the first examples of a molecular dynamics calculation with a continuous potential based on a finite difference time integration method. The calculation for a 500-atom system was performed on an IBM 704, and took about a minute per time step[16, 17].

In 1964, the first simulation using a realistic potential for liquid argon was carried out[13]. In 1967 the phase diagram of argon using the Lennard-Jones potential was calculated, and correlation functions were calculated to test theories of the liquid state[17].

Recently, large amount of work is done on: molecular dynamics simulations of solvated proteins, protein-DNA complexes as well as lipid systems tackling a variety of issues including the thermodynamics of ligand binding and the folding of small proteins. MD simulation techniques are extensively used in experimental procedures such as X-ray crystallography and

NMR structure determination [15].

The recent development in both computer hardware and software allow the performance of realistic simulations of large systems like a protein surrounded by 3000 water molecules[15].

MD applications now exist in many areas sush as liquids, defects, fractures, surfaces, friction, clusters, bimolecular, electronic properties and dynamics, and many others[15].

1.4 Statistical Mechanics

A molecular dynamics simulation gives the positions and velocities of atoms as a function of time. To translate this information into knowledge about macroscopic properties such as pressure, energy, and heat capacities, statistical mechanics is used.

The goal is to understand and predict macroscopic phenomena from the properties of individual molecules making up the system. The system could range from a collection of solvent molecules to a solvated protein. In order to connect the macroscopic system to the microscopic system, time independent statistical averages are often introduced. These averages, corresponding to an experimental observable parameter, are defined in terms of ensemble averages. An ensemble average is an average taken over a large number of replicas of the system considered simultaneously.

These replicas are obtained from a sufficiently long molecular dynamics trajectory. A physical quantity such as pressure is calculated as an arithmetic average of the pressure value at many instances in time along the trajectory. This is possible according to the ergodic hypothesis of statistical mechanics which states that the time average equals the ensemble average.

For example, the average kinetic energy of a system is calculated as shown in equation 1.1[18]

$$K =< K >= \frac{1}{M}\sum_{j=1}^{M}\{\sum_{i=1}^{N}\frac{m_i}{2}v_i v_i\}_j \tag{1.1}$$

where M is the number of configurations in the simulation, N is the number of atoms in the system, m_i is the mass of the particle i and v_i is the velocity of particle i.

A molecular dynamics simulation must be sufficiently long so that enough representative conformations are sampled.

1.5 Classical mechanics

Molecular Dynamics simulations rely heavily on Newton's second law, $F = ma$, where F is the force exerted on the particle, m is its mass and a is its acceleration. Once the force acting on each atom is known, its acceleration, velocity, and position can be easily found as functions of time. The method is deterministic: once the positions and velocities of each atom are known, the state of the system can be calculated at any time in the future or in the past.

Newton's equation of motion is given by

$$F_i = m_i a_i \tag{1.2}$$

where F_i is the force exerted on particle i, m_i is the mass of particle i and a_i is the acceleration of particle i. The force can also be expressed as the gradient of the potential energy,

$$F_i = -\nabla_i V \tag{1.3}$$

Combining these two equations yields

$$-\frac{\partial V}{\partial r_i} = m_i a_i = m_i \frac{\partial^2 r_i}{\partial t^2} \tag{1.4}$$

where V is the potential energy of the system, and r_i is the position of particle i.

To further clarify this concept, a one-dimensional example is discussed:

$$F = m \cdot \frac{dv}{dt} = m \cdot \frac{d^2x}{dt^2} \tag{1.5}$$

Taking the acceleration as constant

$$a = \frac{dv}{dt} \tag{1.6}$$

an expression for the velocity can be easily found by integrating equation 1.6 giving

$$v = at + v_0 \tag{1.7}$$

And since

$$v = \frac{dx}{dt} \tag{1.8}$$

the position x is found by integrating equation 1.8 obtain

$$x = vt + x_0 \tag{1.9}$$

Combining this equation with the expression for the velocity (equation 1.7), we obtain the following relation which gives the value of x at time t as a function of the acceleration, a, the initial position, x_0 , and the initial velocity, v_0.

$$x = at^2 + v_0 t + x_0 \tag{1.10}$$

The acceleration is given as the derivative of the potential energy V (equation 1.4) with respect to the position, r,

$$a = \frac{1}{m}\frac{dV}{dr} \tag{1.11}$$

Thus, to calculate a trajectory, one only needs the initial positions of the atoms, an initial distribution of velocities, and the acceleration, which is determined by the gradient of the potential energy function V [19].

The initial positions can be found from experimental structures, such as the x-ray crystal structure of the protein or the solution structure determined by NMR spectroscopy.

The initial distribution of velocities is obtained by assigning random velocities from a maxwellian distribution at a given temperature, with the added constraint of zero total linear momentum P,

$$P = \sum_{i=1}^{N} m_i v_i = 0 \tag{1.12}$$

The temperature can be calculated from the velocities using the relation [13].

$$T = \frac{1}{3N} \sum_{i=1}^{N} \frac{|p_i|}{2m_i} \tag{1.13}$$

where N is the number of atoms in the system, and p_i is the momentum of particle i.

1.6 Integration Algorithms

The potential energy is a function of the atomic positions ($3N$) of all the atoms in the system. Due to the complicated nature of this function, there is **no analytical solution** to the equations of motion; they must be solved numerically[13].

Numerous numerical algorithms have been developed for integrating the equations of motion. The most widely used are Verlet, The leapfrog, Velocity Verlet, and Beeman's. All these algorithms assume that the positions, velocities and accelerations can be approximated by a Taylor series expansion:

$$\begin{aligned} r(t+\delta t) &= r(t) + v(t)\delta t + \frac{1}{2}a(t)\delta t^2 + ... \qquad (1.14) \\ v(t+\delta t) &= v(t) + a(t)\delta t + \frac{1}{2}b(t)\delta t^2 + ... \qquad (1.15) \\ a(t+\delta t) &= a(t) + b(t)\delta t + \frac{1}{2}c(t)\delta t^2 + ... \qquad (1.16) \end{aligned}$$

where r is the position in three dimensions $3D$, v is the velocity (the first derivative with respect to time), a is the acceleration (the second derivative with respect to time), b is third derivative with respect time, etc.

1.6.1 Verlet

In Verlet algorithm $r(t+\delta t)$ and $r(t-\delta t)$ are expanded using Taylor series

$$\begin{aligned} r(t+\delta t) &= r(t) + v(t)\delta t + \frac{1}{2}a(t)\delta t^2 + ... \qquad (1.17) \\ r(t-\delta t) &= r(t) - v(t)\delta t + \frac{1}{2}a(t)\delta t^2 + ... \qquad (1.18) \end{aligned}$$

Summing these two equations, we get

$$r(t+\delta t) = 2r(t) - r(t-\delta t) + a(t)\delta t^2 \qquad (1.19)$$

The Verlet algorithm uses positions and accelerations at time t and the positions from time $t - \delta t$, to calculate new positions at time $t + \delta t$. The Verlet algorithm uses no explicit velocities. It is straightforward to implement and storage requirements are modest. However, it only offers moderate precision[13].

1.6.2 The leap-Forg

$$\begin{aligned} r(t+\delta t) &= r(t) + v(t - \frac{1}{2}\delta t)\delta t && (1.20) \\ v(t+\frac{1}{2}\delta t) &= v(t - \frac{1}{2}\delta t) + a(t)\delta t && (1.21) \end{aligned}$$

In this algorithm, velocities are calculated at time $t + 1/2\delta t$ and are subsequently used to calculate the positions, r, at time $t + \delta t$. In this way, the velocities leap over the positions, and then the positions leap over the velocities. The advantage of this algorithm is that the velocities are explicitly calculated, however, the disadvantage is that they are not calculated at the same time as the positions. The velocities at time t can be approximated by the relationship[13]:

$$v(t) = \frac{1}{2}[v(t - \frac{1}{2}\delta t) + v(t + \frac{1}{2}\delta t)] \qquad (1.22)$$

This algorithm yields positions, velocities and accelerations at time t.

1.6.3 Velocity Verlet

$$r(t+\delta t) = r(t) + v(t)\delta t + \frac{1}{2}a(t)\delta t^2 \tag{1.23}$$

$$v(t+\delta t) = v(t) + \frac{1}{2}[a(t) + a(t+\delta t)]\delta t \tag{1.24}$$

This algorithm yields positions, velocities and accelerations at time t[13].

1.6.4 Beeman's

This algorithm is closely related to the Verlet algorithm

$$r(t+\delta t) = r(t) + v(t)\delta t + \frac{3}{2}a(t)\delta t^2 - \frac{1}{6}a(t+\delta t)\delta t^2 \tag{1.25}$$

$$\begin{aligned} v(t+\delta t) = {} & v(t) + v(t)\delta t + \frac{1}{3}a(t)\delta t - \frac{5}{6}a(t)\delta t^2 \\ & -\frac{1}{6}a(t-\delta t)\delta t \end{aligned} \tag{1.26}$$

The advantage of this algorithm is that it provides a more accurate expression for the velocities and better energy conservation. The disadvantage is that the more complex expressions make the calculation more expensive[13].

There are three conditions that must be satisfied in any algorithm that integrates equations of motion[13, 20]:

- The algorithm should conserve energy and momentum,

- It should be computationally efficient,
- It should be also allow long time step for integration.

1.7 Molecular Mechanics and Modeling

1.7.1 Introduction

Molecular Mechanics (MM) aims to develop mathematical theories (models) to rationalize and predict experimental observations[21]. It describes molecular structure and properties in a practical manner [22], concentrating on the structural aspects of molecules and not on the electronic aspects.

The Born-Oppenheimer approximation states that the equation of the molecule motions $\Psi(r, R)$ (r is electron position, R is nuclei position) can be separated into two parts: the electrons in the constant field of the fixed nuclei $\Psi(r|R)$ (time-independent Schr$\ddot{o}$dinger equation) and the Newton-like equation of movement for the nuclei $\Theta(R)$(time-dependent Schr$\ddot{o}$dinger equation). [23].

$$\Psi(r, R) = \Psi(r|R)\Theta(R) \tag{1.27}$$

where $\Psi(r|R)$ electronic wavefunction parametrically depends on the ionic position R variables. The equation of motion for the electrons is:

$$(K_e + V)\Psi(r, R) = E_e(R)\Psi(r, R) \tag{1.28}$$

where K_e is kinetic energy of electron, E_e energy eigenvalues of electronic motions, V is the electronic potential.

The equation of motion for nuclei consists of a quantum part [23]

$$(K_e + E_e(R))\Theta(R) = E_T\Theta(R) \tag{1.29}$$

and a classical part

$$m\frac{d^2R}{dt^2} = -\nabla E_e(R) \tag{1.30}$$

These two motions can be studied independently, one by quantum mechanics and the other by molecular mechanics MM [23, 24].

In MM the motions of the nuclei in molecules are studied, while those of the electrons are not[25]. Its range of applicability includes molecules containing thousands of atoms such as proteins, as well as organics, oligonucleotides, peptides, and saccharides. It also deals with the molecule's environment be it vacuum or solvent, in the ground state only. It extends to thermodynamics and kinetic properties as well [22].

Molecular dynamics, conformational energy searching, and docking, require a large number of energy evaluations. MM procedures and methods offer great computational speed allowed by the following principles[22]:

1. Nuclei and electrons are lumped into atom-like particles,

2. Atom-like particles are spherical in shape and have a net charge,

3. Interactions are based on springs and classical potentials,

4. Interactions must be reallocated to specific set of atoms,

5. Interactions determine the spatial distribution of atom-like particles and their energies.

1.7.2 Classical force field component

The classical force field consists of the analytical form of the interatomic potential energy $U = E_e(R)$ as a function of the atomic coordinates of the molecule; and also consist from the parameters that enter U.

Any force field contains the necessary building blocks for the calculation of energy and force[23]:

- A list of atom types as C, O, N, H,etc.
- A list of atomic charges
- Rules for atom-types
- Functional forms of the components of the energy expression
- Parameters for the function terms
- Rules for generating parameters that have not been explicitly defined

- A defined way of assigning functional forms and parameters

1.7.3 The Anatomy of a Molecular Mechanics force field

There are two types of energy terms in the force field: Valence (bonded or internal), and non-Valence terms[26, 27]. The bonded interactions typically include all terms related to the chemical bonds within the molecule such as bond strength, angle bend, and torsion rotation, while the non-Valence interactions include terms that are independent of atomic connectivity such as the Coulomb (electrostatic attraction/ repulsion) and the Van Der Waals interactions[22, 28].

The mechanical molecular model considers atoms as spheres, and bonds as springs that have the ability to stretch, bend, and twist[21, 22]. These simplifying assumptions make the mathematics of atoms (spheres of characteristic radii) much simpler than that with a quantum treatment[22].

Any given conformation of a molecule is associated with a certain amount of energy; and MM aims to predict it. MM energies have no meaning as absolute quantities, but the energy difference between more than two molecules conformation does[22]. A simple and common MM energy expansion equation is given by[22, 27, 21, 28, 26, 23]:

$$Energy = Energy\_valence(Bonded) + Energy(non - Bonded) \quad (1.31)$$

$$Energy = Stretching + Bending + Torsion + (non - Bonded)Interaction$$

(1.32)

The energy equations with parameters that describe the behavior of different kinds of atoms and bonds, are called a force-field or Potential Energy Function[29]. The most common force fields types are AMBER, CHARMM, GROMOS and OPLS/AMBER (Fig 1.2)[27, 20]. Force fields are an intensive area of research under development in a number of laboratories. The mathematical form of the potential energy terms differs from force field to another (Fig 1.2), but the more general and common form potential energy have these energy types:

1. Stretching Energy

   The stretching energy equation is based on Hooke's law [22]. Stretching energy represents the interaction between atomic pairs where atoms are separated by one covalent bond. From IR spectroscopy the relation between bonds and energy is plotted in (Fig1.3) and the stretch atom motion is shown in (Fig1.4) [21]. Stretching energy approximates the energy of a bond as a function of displacement

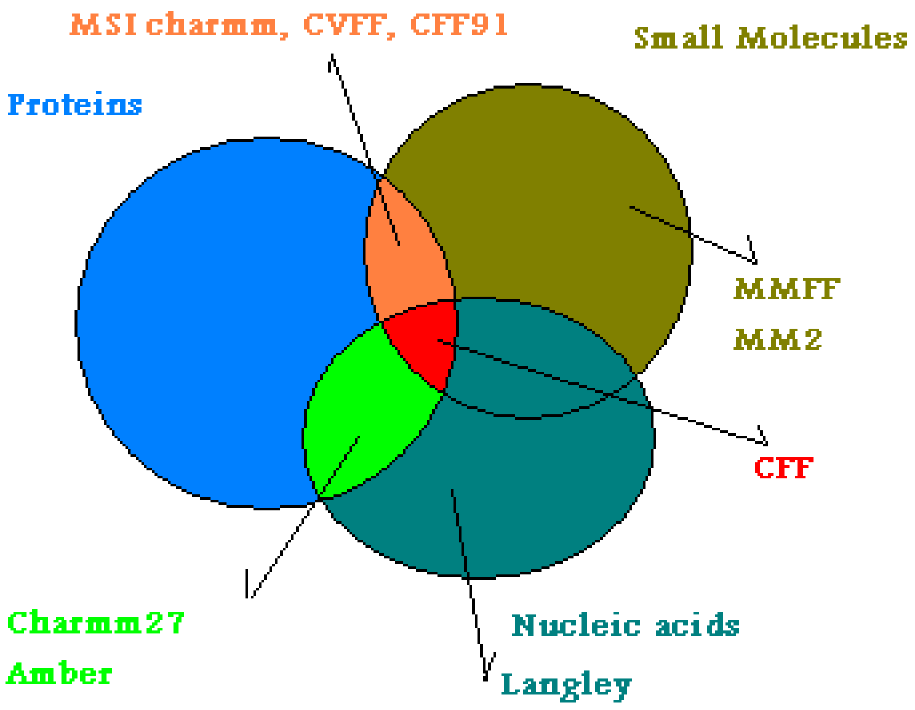


Figure 1.2 Force field type relation. The mathematical form of the potential energy terms differs from force field to another

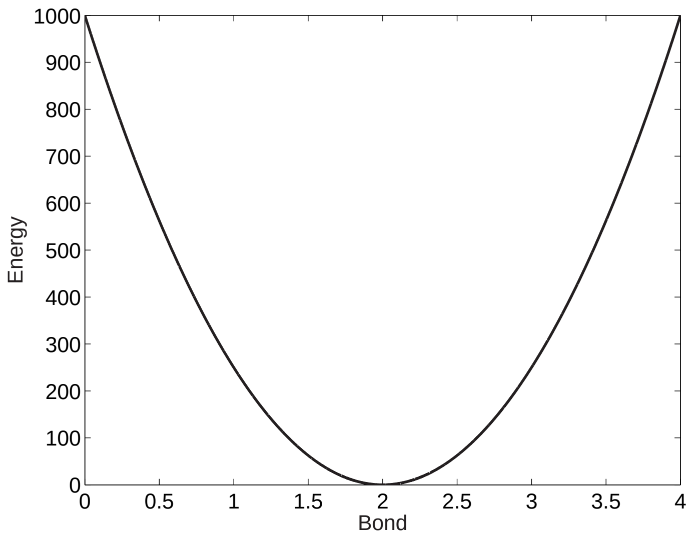


Figure 1.3 The stretching energy

from the ideal bond length equilibrium, r_0. The stiffness. of the bond spring is k_b. Unique k_b and r_0 parameters are assigned to each pair of bonded atoms based on their types as $C-C$, $C-H$, $O-C$, etc[22].

$$E_{bond-stretch} = \sum_{pairs} k_b(r-r_0)^2 \tag{1.33}$$

This equation estimates the energy associated with vibration about the equilibrium bond length (Fig1.3)[23, 27]. This model tends to break down as a bond is stretched toward the point of dissociation. [22, 30].

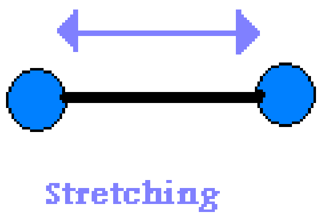


Figure 1.4 The stretching two atoms motion

2. Bending Energy

The bending energy equation is also based on Hooke's law [22]. It is associated with the variation of the bond angles theta θ from the ideal value θ_0, which is also represented by a harmonic potential. From IR spectroscopy the relation between angles and energy is plotted in (Fig1.5) [21]. The bend angle motion is shown in (Fig1.6)

$$E_{bond-bend} = \sum_{angles} k_\theta (\theta - \theta_0)^2 \tag{1.34}$$

Values of θ_0 and k_θ depend on the chemical type of atoms forming the angle [22, 21, 27, 23, 30]. Unique parameters for angle bending are assigned to each bonded triplet of atoms based on their types as $C-C-C$, $C-O-C$, $C-C-H$, etc.

The effect of k_b (stretching) and k_θ (Bending) parameters is to ex-

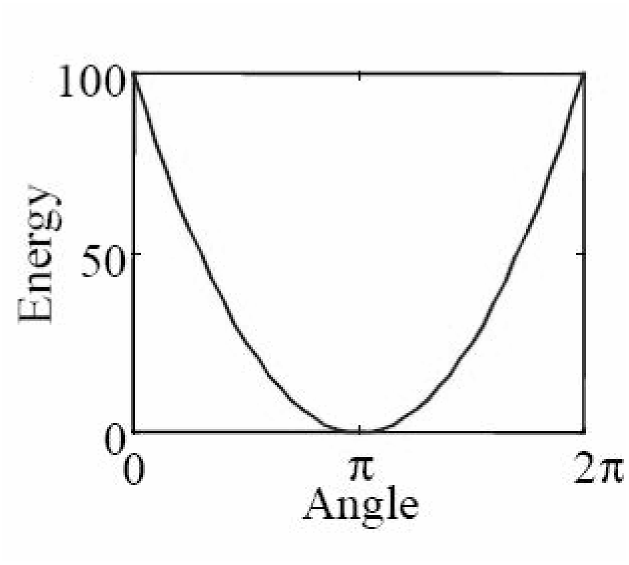


Figure 1.5 bending energy curve

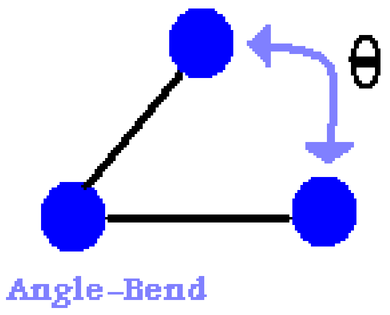


Figure 1.6 bending energy angle

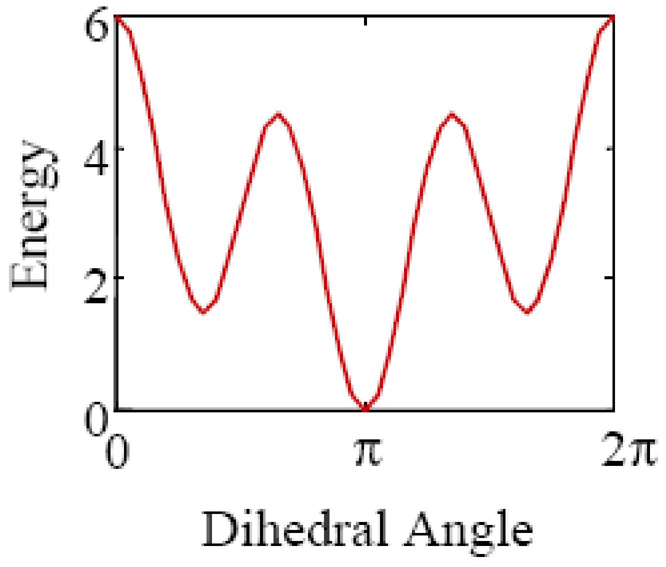


Figure 1.7 Dihedral Energy

pand the slope of the parabola. Larger values mean that more energy is required in deforming an angle or bond from its equilibrium state (Fig1.5) and (Fig 1.6).

3. Torsion Energy (Dihedrals)

   The energy is modeled by a simple sinusoidal periodic function [22]. The torsion angle potential models the presence of steric barriers between atoms separated by 3 covalent bonds (1, 4 pairs) [27]. From IR spectroscopy the relation between bonds and energy is plotted in (Fig 1.7)[21]. The dihedral angle rotation is shown in (Fig 1.8)

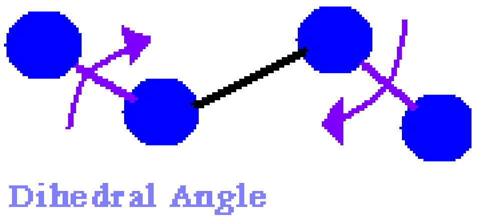


Figure 1.8 Dihedral angle

$$E_{dih} = \sum_{1,4pairs} k_{\phi}(1 + \cos(n\phi - \delta)) \tag{1.35}$$

The k_{ϕ} parameter controls the amplitude of the periodic curve, the n parameter its periodicity, and δ shifts the entire curve along the rotation angle axis ϕ . Unique parameters for Dihedral (torsional rotation) are assigned to each bonded quartet of atoms based on their types as $C-C-C-C$, $C-O-C-N$, $H-C-C-H$, etc (Fig 1.8) [22, 21, 27, 23, 30].

4. Non-bonded Energy Terms

   (a) **Electrostatic energy:** The electrostatic interaction between a pair of atoms is represented by the coulomb potential. [27] The electrostatic energy is a function of the charge on the non-bonded atoms, their interatomic distance r_{ij}, and a molecular dielectric D expression that accounts for the decrease of elec-

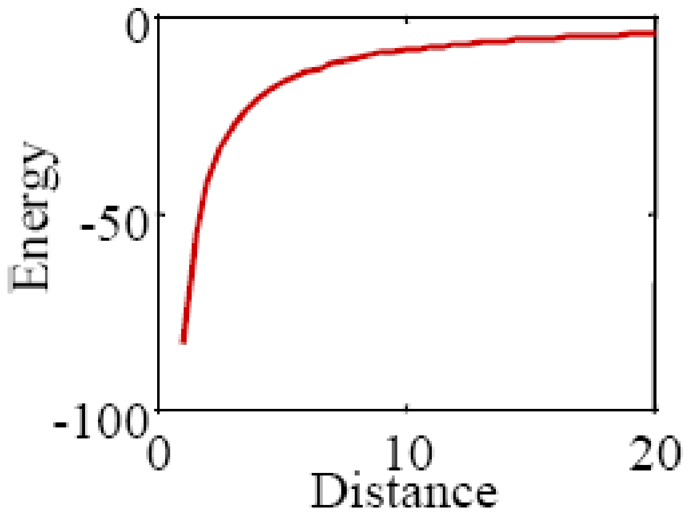


Figure 1.9 The electrostatic potential

trostatic interaction by the environment such as solvent[22]. Electrostatic interaction derived from quantum chemistry, thermodynamics, and empirical schemes. [21] The equation of electrostatic interaction between any two charges q is:

$$E_{electrostatic} = \sum_{non-Bonded(pairs)} \frac{q_i q_j}{D r_{ij}} \tag{1.36}$$

(Fig 1.9). The electrostatic atom partial charge is shown in (Fig 1.10)

(b) **Van der Waals energy** The Van der Waals interaction between two atoms arises from a balance between repulsive and

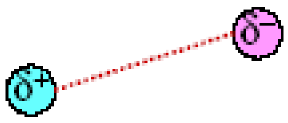


Figure 1.10 The electrostatic atom partial charge

attractive forces[27]. It occurs at short range, and rapidly dies off as the interacting atoms move apart by a few angstroms. Repulsion occurs when the distance between interacting atoms becomes less than the sum of their contact radii[22]. The electrostatic interaction is derived from quantum chemistry, thermodynamics, and empirical schemes. [21] The equation of Van der Waals interaction is (Fig 1.11)[23]:

$$E_{VanderWaals} = \sum_{non-Bonded(pairs)} (\frac{A_{ij}}{r_{ij}^{12}} - \frac{C_{ij}}{r_{ij}^{6}}) \tag{1.37}$$

In addition to these terms some force fields like $CHARMM$ have two extra energy terms. First, the **Urey-Bradley** term that is an interaction based on the distance between atoms separated by two bonds (1, 3 interaction) [27]. The equation of Urey-Bradley is:(Fig 1.12) [23].

$$E_{UB} = \sum k_{UB}(S - S_0)^2 \tag{1.38}$$

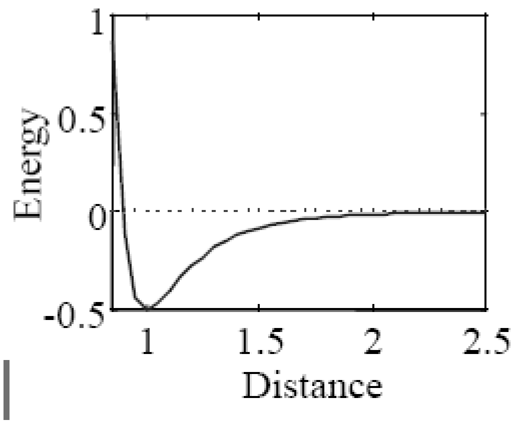


Figure 1.11 Van der Waals interaction energy

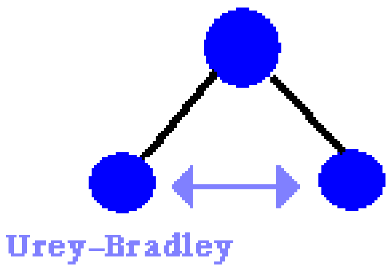


Figure 1.12 Urey-Bradley Energy Potential

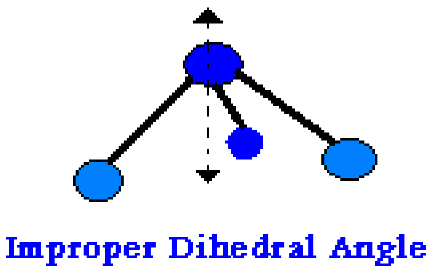


Figure 1.13 improper dihedral energy

The second additional term is the **improper dihedral** term that is used to maintain chirality's and planarity[27]. The equation of improper dihedral energy is:

$$E_\omega = \sum k_\omega (\omega - \omega_0)^2 \tag{1.39}$$

(Fig1.13) [23]. The parameter $k_\omega, k_{UB}, k_\phi, k_\theta$,and k_b are obtained from studies of small model compounds and comparisons to the geometry and vibrational spectra in gas phase. [27].

There are certain limitations to force fields. One of the most important is that no extreme changes in electronic structure are allowed; events such bond making or breaking are not allowed.

Electronic transitions, photon absorption, Electronic transport phenomena, and proton transfer (acid base reaction) cannot be modeled [23, 27]. Some research groups try to mix quantum mechanics and molecular mechanics force fields to model such phenomena.

The power of the force field approach comes from many of its advantages. The force field-based simulations can handle large systems, that are several orders of magnitude faster (and cheaper) than quantum-based calculations. Furthermore, the analysis of the energy contributions can be done at the level of individual or classes of interactions. And finally, the modification of the energy expression to prejudice the calculation [23].

1.8 Essential Dynamics

1.8.1 Background

There are different methods used to investigate the states of protein conformation. One such method is the root mean square deviation ($RMSD$) of $C\alpha$ (alpha carbon) atoms in a protein trajectory for every structure [33]. $RMSD$ describes the "distance" between two (aligned) conformations (protein frames) of a (bio)polymer (or group of selected atoms)[23].

$$RMSD = [< \sum_{N} (r_{1a} - r_{2a}))^2 >]^{1/2} \tag{1.40}$$

where N number of atoms, and the brackets $< ... >$ represent a time average. r_{1a}, r_{2a} are the x, y, z position coordinates of the first and the second conformations for the same atom[8].

$$RMSD = [\frac{1}{N}\sum_{i=1}^{N} < (\Delta x_i)^2 + (\Delta y_i)^2 + (\Delta z_i)^2 >]^{1/2} \tag{1.41}$$

Another method for investigating protein motions in conformational space is "Essential Dynamics"[32, 33]. Essential dynamics, often used under different names such as Principal Component Analysis PCA or quasi-harmonic analysis, is used to analyze the huge data of intermolecular motions produced in MD simulations [1]. Essential dynamics is a tool that extracts large concentrated motions from protein MD trajectories[1, 2], so it is is able to separate concerted motions from the uninteresting local motions[3].

1.8.2 Introduction to ED

Essential Dynamics (ED) has become one of the most interesting and widely used techniques to investigate protein dynamics. ED methods are used to distinguish between the random atom fluctuations from the larger intensive fluctuations [3]. The study of time dependent characteristic of proteins is important for gaining insight into many biological processes and functions such as protein open and close ion channel.

The basic idea of ED is to separate the conformational space into two subspaces: an essential sub-space containing only a few degrees of freedom and a remaining space in which the motions can be considered as physically constrained. The essential degrees of freedom describe motions which are relevant to the function of the protein, while the physically constrained subspace describes irrelevant local fluctuations.

The internal motion of a protein is described by a trajectory $X(t)$, a N-dimensional vector of all atomic coordinates. The essential dynamics method consists of

- Fitting atom trajectories to a reference frame. This step removes overall translational and rotational motions, since only internal motions of the protein are interesting to analyze for protein conformations.
- Constructing a covariance matrix and determining essential motions using the Principal Component Analysis (PCA) technique.

1.8.3 Multivariate covariance and correlation

Multivariate analysis covariance and correlation technique is based on building a matrix of size (nxp) consisting of the variables p and its observation n. In molecular dynamics simulations, the variables are protein atoms and the observations are atomic positions (trajectory) with respect to time. The data is thought of as a set of p n-dimensional vectors of atom trajectories, that is,

$$x_i = [x_{i1}, x_{i2}, ..., x_{in}] \tag{1.42}$$

and so

$$X = [x_1, x_2, ..., x_p] \tag{1.43}$$

where, x_j is the trajectory vector containing values of one of the atoms for all n positions through time. We can define[34]:

- The mean $\overline{x}_1$, taken to be the mean of each variable over all the positions as

  $$\overline{X} = [\overline{x}_1, \overline{x}_2, ..., \overline{x}_p] \tag{1.44}$$

  where $\overline{X}$ is mean total matrix.

- Sample covariance (measure of linear association between pairs of atoms i and j) given by the matrix

  $$Cov(X) = S \tag{1.45}$$

  with elements

$$s_{ij} = \frac{1}{n-1}\sum_{k=1}^{n}(x_{ik} - \overline{x}_i)(x_{jk} - \overline{x}_j) \quad i = 1, .., p \quad j = 1, .., p \tag{1.46}$$

A negative covariance implies that most trajectory positions with large values for atom i have small values for atom j or vice versa. A positive covariance implies that either large or small values are obtained for both atoms, and zero covariance implies that there is no particular association between the variables over the whole trajectory[34]. The sample variance is given by determinant of covariance matrix S.

1.8.4 Principal component analysis PCA

PCA is a way of identifying patterns in data, and expressing the data in such a way in order to highlight the similarities and differences in that data. PCA is a powerful tool for analyzing data because of its ability to recognize patterns and compress the data. This means that PCA can reduce the number of dimensions without much loss of information[35].

The steps needed to perform a Principal component analysis PCA on a set of data are [35]:

- Get the Data.
- Subtract the mean (average) from each of data dimensions,
- Calculate the covariance matrix by taking the dot product between each column with the other.
- Calculate the eigenvectors and eigenvalues of the covariance matrix.

- Choosing suitable components and forming a *feature vector*, reduce the dimension of data and compress it. Choose in the feature vector the eigenvectors with largest eigenvalue (components).

- Derive the new data set by taking the transpose of the *feature vector* and multiplying it with the original data set.

The objectives of Principal Components Analysis are[34, 36].

- Dimensionality reduction, helping the understanding of a large data set.

- Determining of linear combinations of variables, that can be thought of as a new set of axes that have been rotated from the original axes to fit the data better (find important relationships between variables by using smaller set).

- Choosing of the most useful variables; finding a smaller sets of base vectors that still describe the data well.

- Visualization of multidimensional data by decomposing the original data into sets of distinct patterns over time points that can be recombined to create the original data. Find most common patterns in the data and visualize it in reduced component space.

- Identification of fundamental variables, this allows the examination of the influence of a given variable on the most influential components.

- Identification of groups of objects.

The trajectories set that are obtained from the MD simulation are organized into (nxp) matrix, where p is the atoms (variables), and n is the observation frames of moving trajectories with respect to time (atom positions). The covariance matrix of the atoms trajectory is the (pxp) matrix. $S = Cov(X)$, from equation 1.50 the elements of matrix S are constructed from the deviation of the j^{th} coordinate from its column average at a time i [34]. To find the principal component of any data we search for p uncorrelated, linear combination of vectors $x_1, x_2, ...x_p$ with maximum variance [34] Equation 1.43.

We search for p-dimensional vector a_i that gives the p principal components

$$\begin{aligned} y_1 &= a_{11}x(1) + a_{21}x(2) + ... + a_{p1}x(p) = a_1X \qquad (1.47) \\ y_2 &= a_{12}x(1) + a_{22}x(2) + ... + a_{p2}x(p) = a_2X \\ &\vdots \\ y_p &= a_{1p}x(1) + a_{2p}x(2) + ... + a_{pp}x(p) = a_pX \qquad (1.48) \end{aligned}$$

This system of p linear combinations where a_{ij} is scalar constant, the variance of the i^{th} linear combination is.

$$Var(y_i) = Var(a_i x) = a_i Var(x) a_i \tag{1.49}$$

Those variances in equation 1.47 are maximized. The variance of the i^{th} principal components is

$$Var(y_i) = a_i S a_i \tag{1.50}$$

Subject to the constrain $a_i a_i = 1$, since the variance increases by increasing a_i[34]. The principal components are found by maximizing $a_i S a_i$ subject to $a_i a_i = 1$. The results are the coefficients of the principal components which are the eigenvector of the covariance matrix, so equation 1.47 became:

$$\begin{aligned} y_1 &= e_{11}x(1) + e_{21}x(2) + ... + e_{p1}x(p) = e_1 X \qquad (1.51) \\ y_2 &= e_{12}x(1) + e_{22}x(2) + ... + e_{p2}x(p) = e_2 X \\ &\vdots \\ y_p &= e_{1p}x(1) + e_{2p}x(2) + ... + e_{pp}x(p) = e_p X \qquad (1.52) \end{aligned}$$

Where e_i is the eigenvector corresponding to the i^{th} largest eigenvalue, λ_i, of the covariance matrix S. The spectral decomposition of S is:

$$S = \sum_{i=1}^{p} \lambda_i e_i e_i \Rightarrow e_i S e_i = \lambda_i \tag{1.53}$$

From equation 1.50 the variance of the i^{th} principal component is its eigenvalue, λ_i, and the total variance is the summation of all eigenvalues of each principal component. From that we can calculate the proportion of each principal component (its rank) with respect to total variance. These eigenvectors of the covariance matrix or the coefficients of principal components provide us with the correlation information between the k^{th} atom trajectory and the i^{th} principal component[34].

1.8.5 Essential Dynamics Technique for Protein

There are some steps for the essential dynamics method:

- Fitting and aligning atom trajectories to a reference frame. This step removes overall translational and rotational motions, since only internal motions of the protein are interesting to analyze. A simple procedure (Kabsch, 1976) was used to find the best rotation and translation for a given vector set into another vector by minimizing the weighted sum of squared deviations[37, 38, 3].

- Constructing a covariance matrix and determining essential motions using Principal Component Analysis PCA (see section 1.8.4).

- Calculating the cross-correlation matrices for the $C\alpha$ atoms. It is possible to calculate the matrices using either the Cartesian displacements of each of the x,y,z coordinates (of the $C\alpha$ atoms), or, by taking the dot product of the atomic displacements from their average values.

- Calculating of the corresponding eigenvalues and eigenvectors by Single Value Decomposition (SVD) method.
- Calculating the projection of the atomic fluctuations on chosen eigenvectors.
- Projecting of the trajectory on selected (significant) eigenvectors with larger eigenvalue.
- Constructing an artificial trajectory depicting the motion due to a single (selected) eigenvector.
- Projecting the trajectory on selected (significant) eigenvectors from different parts of the simulation in order to show the similarities of the eigensates.

Fiting and Aligning the structure

To study the internal atom motions of a protein requires the removal of all translation and rotation motions from its trajectory[37, 3, 2, 32]. Finding the transformation matrix that fits and aligns the atoms of one protein structure x (frame) to another protein structure y (frame) in an optimal way can be done by the following procedure [37]. Let x_n and y_n (n=1,2,...,N (Number of atoms)) two configration structure (frames), and w_n be the weight corresponding to each pair of atoms for x_n frame and y_n frame.

- compute $3X3$ matrix R with components

  $$r_{ij} = \sum_{n=1}^{N} (w(n) * (y_{n,i} - comy_i) * (x_{n,j} - comx_j)) \tag{1.54}$$

  where i,j take values from one to three to represant the atom trajectory components x, y, z. comy is the centroid of the y_n frame, and $comy_i$ is the x center component of y_n frame if $i = 1$.

- Form $R^T R$ matrix.

- Find the eigenvalues and eigenvectors of $R^T R$ in a matrix.

- Transpose the eigenvector matrix a to put the vectors in rows

- Sort the eigenvalues and its corresponding eigenvectors from largest to smallest

- Determine

  $$b_i = \sum_{j=1}^{3} R_{ij} * a_{ij} \tag{1.55}$$

- Compute U that minimize the *rms*

$$u_{ij} = \sum_{k}^{3}(b_{ki} * a_{kj}) \tag{1.56}$$

Check the determinant of U. If it's negative, we need to flip the sign of the last row. U is the rotation matrix that takes one protein structure to another[appendix A.1].

Covariance matrix

We want to investigate a trajectory of protein through time in terms of the relationship between individual variables (atoms and their position through time)[39]. The mean tells us where the middle point of the data is, but not more. $\overline{X} = \sum(x_i)/n$ where x_i refers to an individual number in the data set, n is the number of elements in the data set, and $\overline{X}$ "X bar" indicates the mean of the set. The mean does not provide us with any information about the spread of the data. The standard deviation s (SD) of a data set measures the spread of the data around the mean. So it is "the average distance from the mean of the data set to a point".

$$s = \sqrt{\frac{(\sum(X_i - \overline{X}))^2}{(n-1)}} \tag{1.57}$$

Variance also measures the spread of data in the data set. It is identical to the standard deviation s. Variance is the square of the standard deviation s^2. Standard Deviation and variance just measure the spread of data in one dimension but covariance measures the spread of data and the relationship between the dimensions if they exist in the data sets[39]. The formula of covariance is very similar to the variance formula[39].

$$Var = \frac{\sum((X_i - \overline{X})(X_i - \overline{X}))}{(n-1)} \tag{1.58}$$

$$Cov = \frac{\sum((X_i - \overline{X})(Y_i - \overline{Y}))}{(n-1)} \tag{1.59}$$

The covariance always measures the relationship correlation between two dimensions of a data set, but in MD trajectory we have 3N (N number of atoms) dimensions. There is more than one covariance measurement that needs to be calculated. For example, for three dimensions x, y, and z we need to calculate Cov(x,y), Cov(x,z), Cov(y,z). This means we need to calculate $\frac{n!}{(n-2)!*2}$ covariances plus n variance value or $\frac{n(n+1)}{2}$.

For efficient use, and to deal with this calculation, the covariance values are arranged in a matrix called the covariance matrix with coefficients

$$C_{ij} = \sum(x_i - \overline{x}) * (y_i - \overline{y}) \tag{1.60}$$

Each entry in the covariance matrix is the result of calculating the covariance between two separate dimensions.

The diagonal elements of the covariance matrix ($i = j$) correspond to the mean-square (m.s.) fluctuations of atom i. They can be converted to simulated atomic B-factors, B_i, using the relation[40, 27]:

$$B_i = (\frac{8\pi^2}{3} C_{ii}) \tag{1.61}$$

B-factor describes the reduction of the intensity of Bragg scattering due the motion of atoms about their equilibrium position.

For a 20ns long MD trajectory containing 57 $C\alpha$ atoms, you need to calculate the averages for 3N dimensions by using for each dimension 400,000 different values from each frame in the course of trajectory. Then save the deviation from each dimension average in new feature data matrix with dimension of $(400,000 \times 171)$ for water and other one for vacuum. After that find the dot product of each column with the other, which means the covariance between these two dimensions.

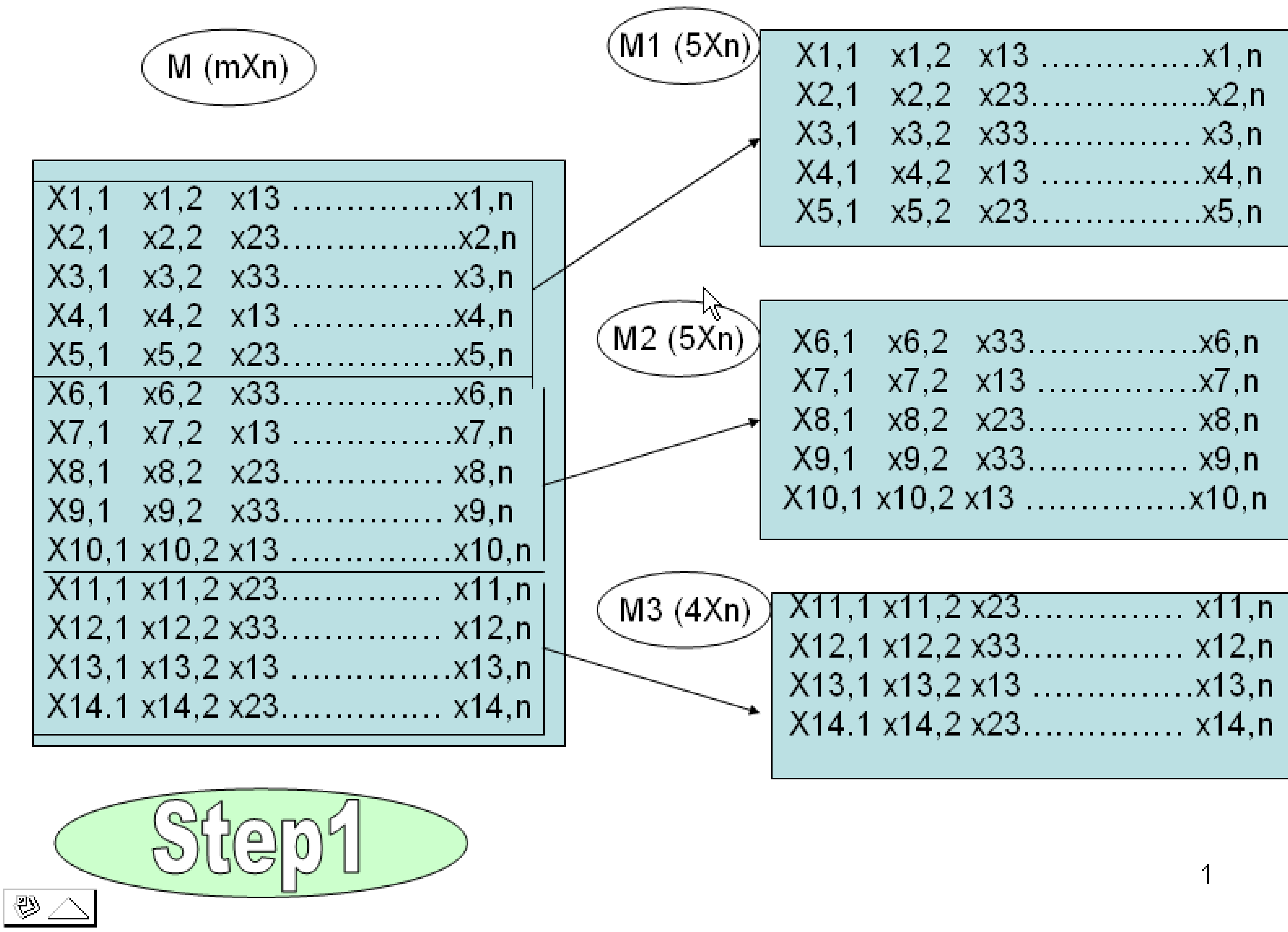
If more than 400000 structure for simulation of 20 ns are used in the essential dynamics analysis, it becomes computationally impossible to build the corresponding covariance matrix (on the average PIV workstation with about 256 MB of memory). Both CPU time and memory needed to build the covariance matrix from all structures increase rapidly with the size of the system (approximately proportional to N^2)[2].

It is possible, however, to overcome this problem by dividing the overall structure matrix to substructure matrices because each part of the structures adds new aspects and meaning to the covariance matrix. The procedure is as follows:

- Divide the structure matrice of 400000×171 into substructure matrices of smaller size that can fit in the computer memory (1ns 20000×171) (Fig1.14).
- For each substructure matrix, find the sum and number of rows as (Fig1.15).
- Find the average for the all structure matrix by add the summation of step 2 and divide it by the total number of rows (Fig1.16).
- Find the deviation of each column's structure (frame) from the column averages from step 3 separately (Fig1.17).
- Find the dot product between every two columns (covariance) in each substructure matrix separately, and save them in a subcovariance square matrix with dimension number of columns.
- Add these subcovariance matrcies to get the total covariance matrix of overall structures.

The mathematical proof of the above method for covariance matrix is as follows:

$$C_{ij} = \sum_{n=1}^{n=N} (X_{in} - < X_i >)(X_{jn} - < X_j >) \tag{1.62}$$



M (mXn)
X1,1 x1,2 x13 ....x1,n
X2,1 x2,2 x23.................x2,n
X3,1 x3,2 x33............... x3,n
X4,1 x4,2 x13 ....x4,n
X5,1 x5,2 x23...............x5,n
X6,1 x6,2 x33...............x6,n
X7,1 x7,2 x13 ....x7,n
X8,1 x8,2 x23............... x8,n
X9,1 x9,2 x33............... x9,n
X10,1 x10,2 x13 ....x10,n
X11,1 x11,2 x23.............. x11,n
X12,1 x12,2 x33.............. x12,n
X13,1 x13,2 x13 ....x13,n
X14.1 x14,2 x23.............. x14,n
M1 (5Xn)
M2 (5Xn)
M3 (4Xn)
Step1
1

Figure 1.14 Divide the structure overall matrix M into substructure matrices M1, M2, etc, of smaller size that can fit in computer memory

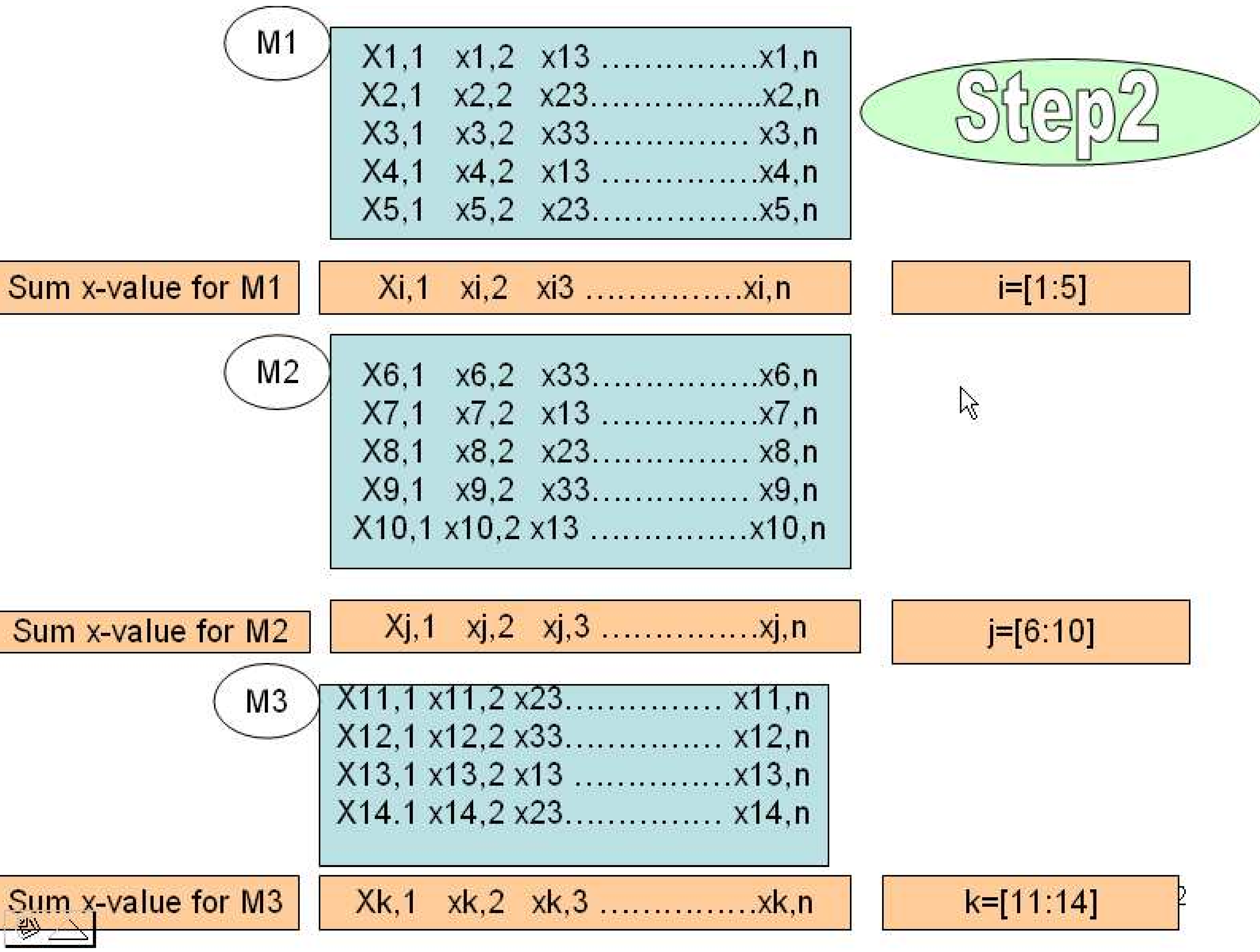

M1
X1,1 x1,2 x13 ....x1,n
X2,1 x2,2 x23................x2,n
X3,1 x3,2 x33.............. x3,n
X4,1 x4,2 x13 ....x4,n
X5,1 x5,2 x23..............x5,n
Step2
Sum x-value for M1
Xi,1 xi,2 xi3 ....xi,n
i=[1:5]
M2
X6,1 x6,2 x33..............x6,n
X7,1 x7,2 x13 ....x7,n
X8,1 x8,2 x23.............. x8,n
X9,1 x9,2 x33.............. x9,n
X10,1 x10,2 x13 ....x10,n
Sum x-value for M2
Xj,1 xj,2 xj,3 ....xj,n
j=[6:10]
M3
X11,1 x11,2 x23.............. x11,n
X12,1 x12,2 x33.............. x12,n
X13,1 x13,2 x13 ....x13,n
X14.1 x14,2 x23.............. x14,n
Sum x-value for M3
Xk,1 xk,2 xk,3 ....xk,n
k=[11:14]

Figure 1.15 For each substructure matrix M1, M2, etc, find the sum and number of rows in each one

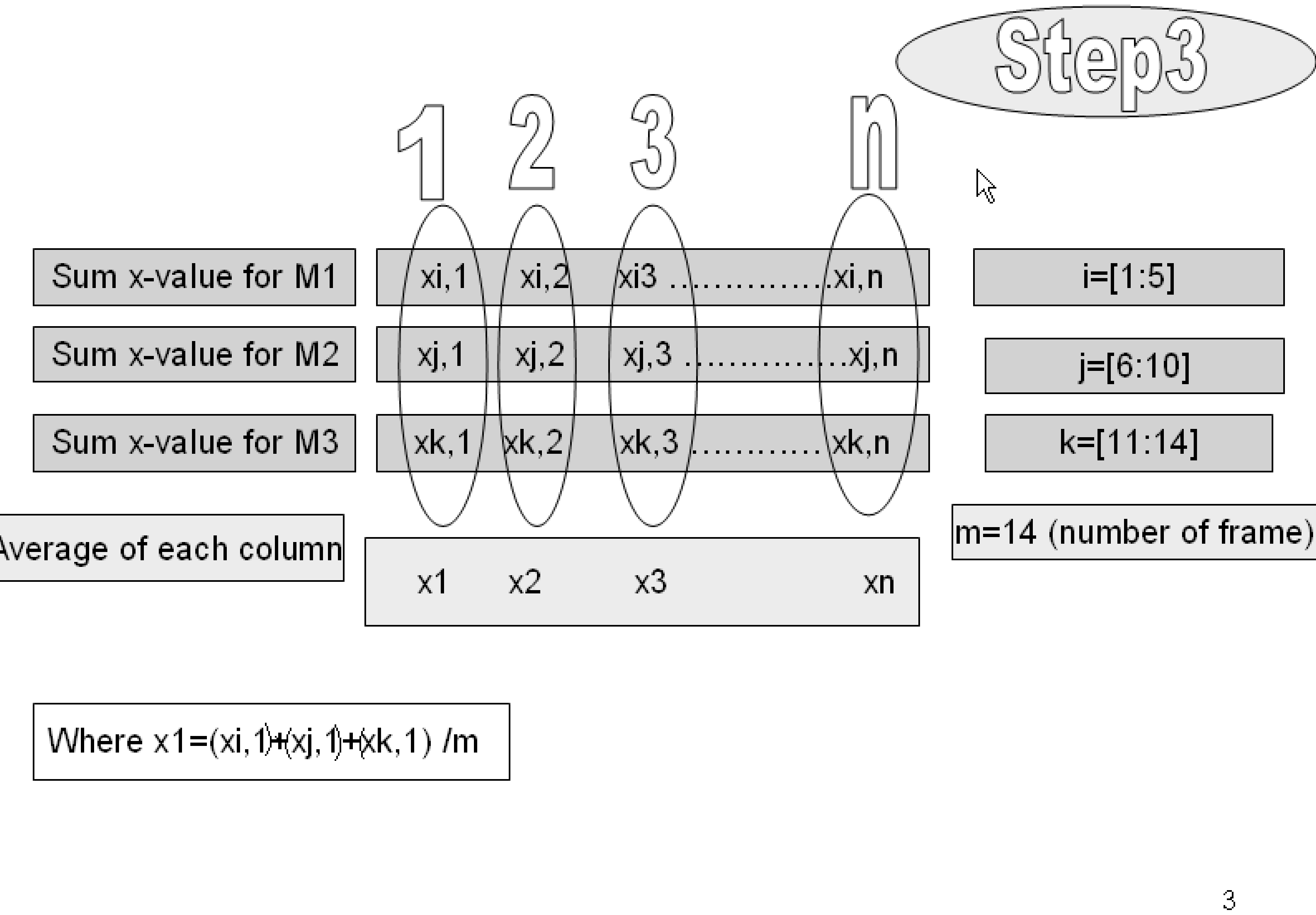
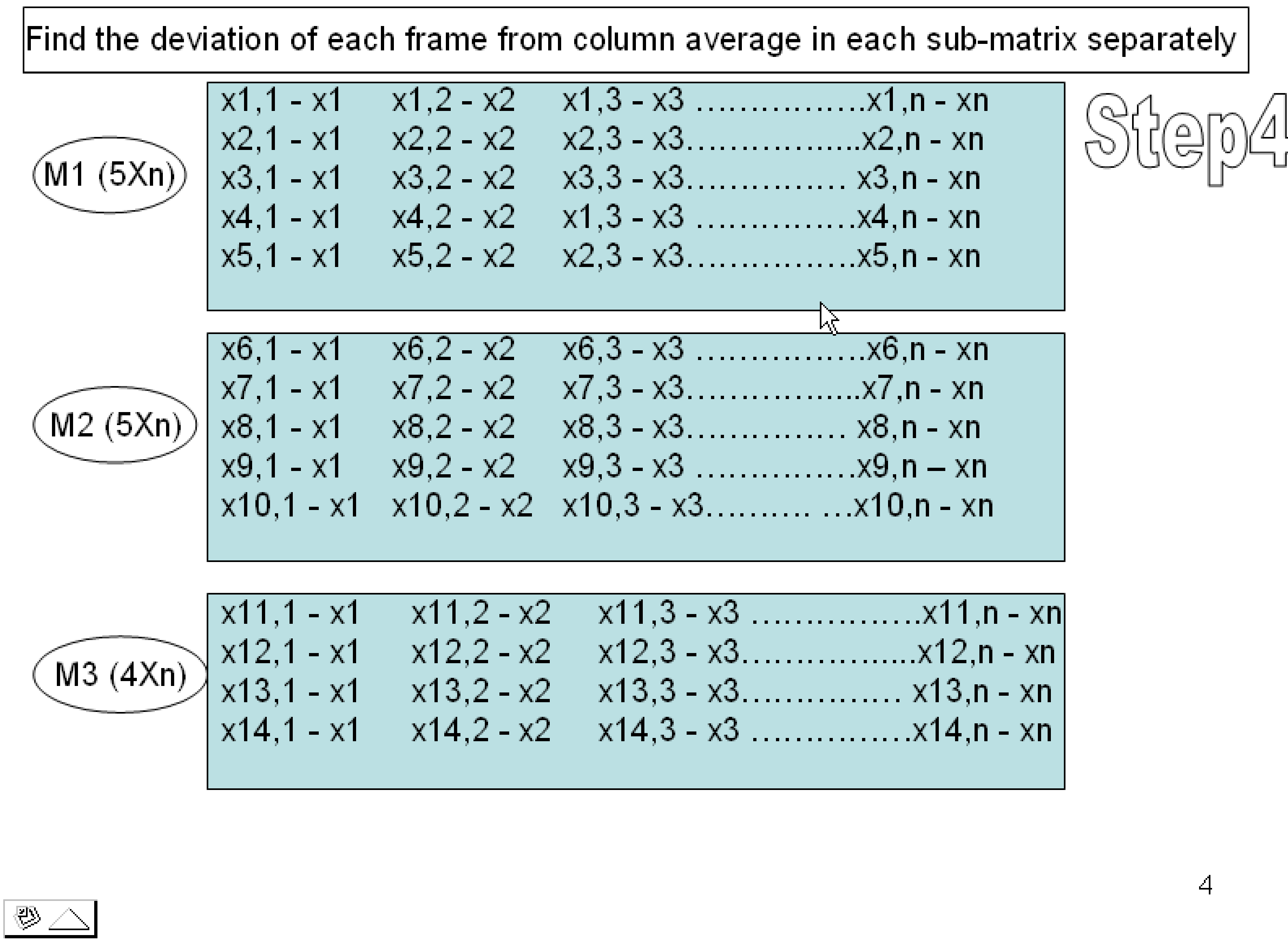


Figure 1.16 Find the average for the all structure column as X1, X2, etc, and divide it on total number of rows



Find the deviation of each frame from column average in each sub-matrix separately

Step4

M1 (5Xn)

x1,1 - x1 x1,2 - x2 x1,3 - x3 …………….x1,n - xn
x2,1 - x1 x2,2 - x2 x2,3 - x3……………..x2,n - xn
x3,1 - x1 x3,2 - x2 x3,3 - x3…………… x3,n - xn
x4,1 - x1 x4,2 - x2 x1,3 - x3 ……………x4,n - xn
x5,1 - x1 x5,2 - x2 x2,3 - x3……………x5,n - xn

M2 (5Xn)

x6,1 - x1 x6,2 - x2 x6,3 - x3 …………….x6,n - xn
x7,1 - x1 x7,2 - x2 x7,3 - x3……………..x7,n - xn
x8,1 - x1 x8,2 - x2 x8,3 - x3…………… x8,n - xn
x9,1 - x1 x9,2 - x2 x9,3 - x3 ……………x9,n – xn
x10,1 - x1 x10,2 - x2 x10,3 - x3……… …x10,n - xn

M3 (4Xn)

x11,1 - x1 x11,2 - x2 x11,3 - x3 …………….x11,n - xn
x12,1 - x1 x12,2 - x2 x12,3 - x3……………..x12,n - xn
x13,1 - x1 x13,2 - x2 x13,3 - x3…………… x13,n - xn
x14,1 - x1 x14,2 - x2 x14,3 - x3 ……………x14,n - xn

4

Figure 1.17 Find the deviation of each column's structure

Dividing the X matrix with the same number of columns and different number of rows is equivalent to dividing the covariance matrix summation in equation 1.62 as

$$C_{ij} = \sum_{n=1}^{n=M} (X_{in} - < X_i >)(X_{jn} - < X_j >) + \sum_{n=M}^{n=M1} (X_{in} - < X_i >)(X_{jn} - < X_j >) + .. \tag{1.63}$$

so that the covariance matrix will be

$$C_{ij} = \sum_{i=1}^{i=(N/M)} \sum_{n=1}^{n=M} (X_{in} - < X_i >)(X_{jn} - < X_j >) \tag{1.64}$$

1.9 Molecular and Essential Dynamics Tools used

1.9.1 *NAMD*: MOLECULAR DYNAMICS SIMULATION

NAMD was published by the Theoretical Biophysics Group at the University of Illinois, it is an object-oriented molecular dynamics code for a parallel system consisting of tens of processors. *NAMD* was designed for high-performance simulation of large bimolecular systems and distributed free of charge along with the source code [33]. The *NAMD* computational scientific tool exist for all hardware platforms. The documentation, features, training, and news are available from http://www.ks.uiuc.edu/Research/namd/. *NAMD* can run under individual windows or UNIX workstations, as well as on clusters of workstation networks.

NAMD is funded by the National Institutes of Health (*NIH*), which is National Center for Research Resources for Macromolecular Modeling and Bioinformatics [33].

$NAMD$ computes atomic trajectories by solving equations of classical mechanics (equations of motion) numerically. Realistic simulation of systems as large as an enzyme surrounded by 30000 water molecules can be performed.

$NAMD$ has many advantages over the previous molecular dynamics simulation packages, such as $X-PLOR$[41] and $CHARMM$[27], the most important difference is that the previous tools were developed for serial machines.

Other advantages for $NAMD$ are:

- Force Field Compatibility: $NAMD$ can accept many force force field types.
- Efficient full electrostatics algorithms that reduce the computational complexity from $(N * N)$ to $(N * log(N))$.
- $NAMD$ uses a multiple time step algorithm.
- Compatibility in input and output file formats with other tools.
- Dynamics simulation options: It can carry several options such as constant energy and temperature.

- $NAMD$ is easy to modify and extend because it was designed using object-oriented style and written in $C++$.

$NAMD$ uses a set of files to build the molecular system simulation environment. These include input, output, and configuration parameter files that have special formats. These requirement files supply $NAMD$ with all the information needed in order to performe molecular dynamics simulation.

1.9.2 VMD:Visual Molecular Dynamics: Computer Tool

VMD is a Molecular Visualization program designed for the interactive visualization and analysis of a molecular dynamics simulation. VMD uses an object-oriented design, and is written in C++. VMD was developed by the theoretical biophysics group at the University of Illinois and the Beckman Institute [42].

Some features of VMD include:

- General molecular visualization tools to display molecules containing any number of atoms.

- It can read most of the formate files like PDB, and DCD. It can also convert from one format to another automatically, using the scientific tool Babel[43]. It can display the structure in various ways including licorice, ribbons, van der Waals, spheres, and molecular surfaces[43].

- Rendering the molecular structure directly to many file formats such as a postscript , ray tracing programs format, Raster3D, $POV4$, Rayshad, jpeg, etc.

- VMD support for several input and output devices other than the usual monitor, keyboard, and mouse. These include magnetic, trackers, haptic feedback devices, wands, dial boxes, etc. These device are used to get position and orientation information of the molecular systems.

- It understands Tcl scripts language[44] and uses it to perform molecular analysis commands. These commands include methods to extract information about a set of atoms and molecules, vector and matrix routines for coordinate manipulation, and functions for computing values like the center of mass, radius of gyration, and root mean square deviation $rmsd$.

1.9.3 Unix Cat Trajectory Binary File (Catdcd)

Before get into CatDCD tool it is important to describe DCD trajectory formate file. DCD is a binary formate file divided into two parts. The header that contains information like number of atoms, number of frame, time step between frames , date of modification, formate type (charmm, x-plor)...etc. These things exist in three main blocks, each block begin with integer number (four byte in length) and end with the same integer thats show how many bytes exist into this block. In each place in the block exist specific data type as "COOR" after the first integer.

The rest of the *DCD* file is three binary blocks of the trajectory positions or velocities of each atom in the frame until the end of all frames(Appendix B.7).

Catdcd functions much like the Unix "cat" command [45]: it concatenates trajectory DCD files into a single DCD file. It also allows the specification of atoms and frames that need to be written into the output file. Thus DCD's can be split up as well as combined to form trajectory files.

1.9.4 Byte Order Reverser Flipdcd

FlipDCD is a small utility tool distributed with the *NAMD* package [46]. It is used to reverse the endianism (byte ordering) of binary DCD trajectory files of *Charmm*, and *NAMD*. This can be useful when running simulations on one architecture and visualizing or analyzing the results on another hardware architecture.

Platform such as windows or unix use different byte orders. Unix uses little endianism and windows uses big endianism.

FlipDCD provides a mechanism for converting the endianism (byte ordering) of $CHARMM$, $X - PLOR$, and $NAMD$ DCD trajectory files. DCD files may be loaded by visualization and analysis programs on platforms with the opposite byte ordering of the platform on which they were originally generated. This allows one to use a Windows PC to read DCD trajectories generated on a Sun or an unix and allows a Sun or an windows to read trajectory files produced on a PC cluster running Linux. FlipDCD does the endianism conversion by memory mapping the DCD file[46], and converting the endianism in-place. This provides a relatively high performance method to perform endianism conversion.

1.9.5 matlab and its interrupter Octave

Matlab (Matrix laboratory) is an interactive software system for numerical computations and graphics[47]. Matlab is especially designed for matrix computations: solving systems of linear equations, computing eigenvalues and eigenvectors, factoring matrices, and so forth. In addition, it has a variety of graphical capabilities, and can be extended through programs written in its own programming language[47].

Many such programs come with the system; a number of these extend matlab's capabilities to nonlinear problems, such as the solution of initial value problems for ordinary differential equations. Matlab is designed to solve problems numerically, that is, in finite precision arithmetic. Therefore it produces approximate rather than exact solutions, and should not be confused with a symbolic computation system (SCS)[47] such as Mathematica or Maple. It should be understood that this does not make Matlab better or worse than an SCS; it is a tool designed for different tasks and is therefore not directly comparable.

Matlab performs many matrix computations. Among the most useful is the computation of eigenvalues and eigenvectors with the *eig* command. If A is a square matrix, then $ev = eig(A)$ returns the eigenvalues of A in a vector, while $[V, D] = eig(A)$ returns the spectral decomposition of A: V is a matrix whose columns are eigenvectors of A, while D is a diagonal matrix whose diagonal entries are eigenvalues of A[47].

1.9.6 Principal Component Analysis Tool (CARMA)

CARMA is a molecular dynamics principal component analysis and distance map calculator. It can do most essential dynamics analysis with some limitations[38]. It supports most of the steps required for a principal component analysis of molecular dynamics trajectories.

Carma can [38]:

- Read a PDB file, or, a DCD (trajectory) and its corresponding PSF file, and produce postscript file(s) containing a grayscale representation of the corresponding $C\alpha$-$C\alpha$ distance map(s).

- Read a DCD (and its PSF) file and calculate the average $C\alpha$-$C\alpha$ distance map over all defined frames plus an additional map containing the root mean square deviation of these $C\alpha$-$C\alpha$ distances from their average values (over the same set of frames).

- read a DCD (and its PSF) file and perform most of the steps involved in a principal component analysis of the trajectory ($C\alpha$ atoms only).

Chapter 2
Results and Analysis

2.1 Molecular and Essential Dynamics Method used

Following is a recipe of how to perform a MD simulation of $BPTI$ using $NAMD$:

Four files should be supplied:

- Protein Data Bank (PDB), file which stores atomic coordinates and/or velocities for the molecular system. PDB files may be generated by hand, but they are also available via the Internet for many proteins at http://www.pdb.org (Appendix B.1).

- Protein Structure File (PSF), which stores structural information of the protein, such as various types of bonding interactions(Appendix B.2).

- A force field parameter file. A force field is a mathematical expression of the potential which the atoms in the system feel. CHARMM, X-PLOR, AMBER, and GROMACS are four types of force fields, and $NAMD$ is able to use any one of them. The parameter file sets bond strengths, equilibrium lengths, etc(Appendix B.3).

- A configuration file, in which the user sets all the options that

$NAMD$ should adopt in running a simulation. The configuration file tells $NAMD$ how the simulation is to be run(Appendix B.4).

There are six steps in any typical MD simulation (Fig2.1):

- Prepare molecule: get *pdb* and psf files for your molecular system,
- Minimization: reconcile observed structure with force field used at zero temperature.
- Heating: raise the temperature of the system to the desired one.
- Equilibration: Ensure system is stable by monitoring system temperature.
- Dynamics:(Production phase)simulate under certain condition then collect your data.
- Analysis: evaluate the observable macroscopic level properties for your collected data.

2.1.1 Preparation of the Protein Molecule

This step involves generating the $NAMD$ required files to represent $BPTI$ protein. Steps:

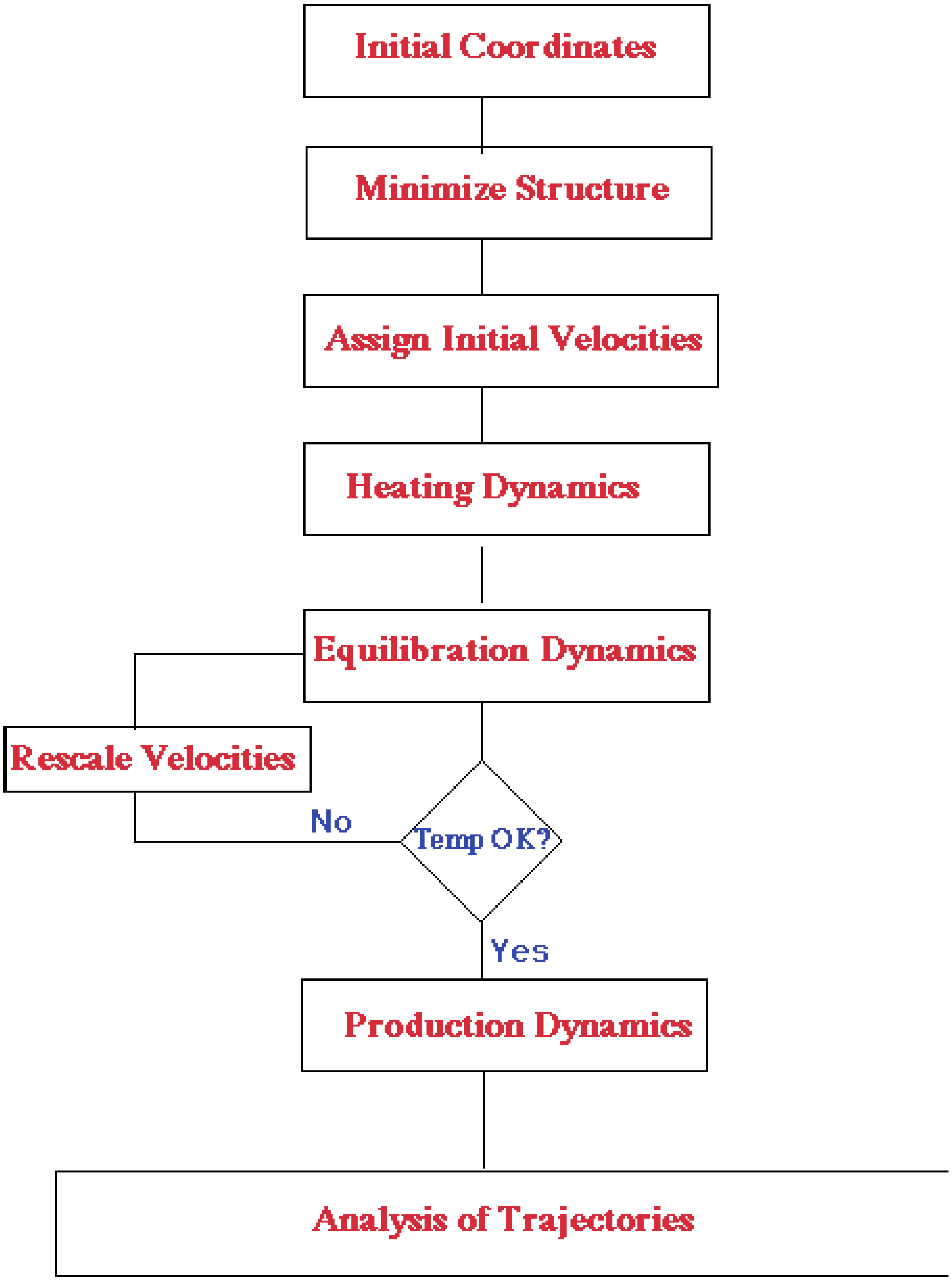


Figure 2.1 Molecular Dynamics technique, MD simulation phases

1. A PDB of $BPTI$ is available through the internet at www.rcsb.org (Appendix A.1) format]. The X-ray structure of 6PTI.pdb was obtained from the Protein Data Bank (PDB). This structure does not contain the hydrogen atoms of BPTI. Some modification on 6PTI.pdb is required by adding hydrogen atoms. The PDB file that will be generated with the PSF will contain guessed coordinates for hydrogen atoms of the structure. The minimization step in MD simulation will ensure hydrogen atom positions are reasonable.

2. A PSF of $BPTI$ must be created from the initial PDB ($6PTI$), parameter, and topology files(Appendix B.6)[48]. Topology files available through internet at
   http://www.pharmacy.umaryland.edu/faculty/amackere/research.html.

Parameter files contain the force constants necessary to describe the bond, angle, torsion energy, and non-bonded interaction (Van der Waals and electrostatics). Parameter files also suggest parameters for setting up the energy calculations (Appendix B.5). Topology files contain atom types that are assigned to identify different elements and different molecular orbital environments[23], in addition to the charge that is assigned to each atom. The topology file contains the connectivity between atoms (Appendix B.6) that are available at http://www.pharmacy.umaryland.edu/faculty/amackere/research.html.

To build the PSF and the modified PDB files for *BPTI*:

1. Split the input file 6*PTI.pdb* into two segments one called 6*PTI_protein.pdb* and the other called 6*PTI_water.pdb* using the grip UNIX command[50].

2. Run *VMD* in text mode and type the following lines in the prompt of *VMD*

   (a) **package require psfgen**

   Runs psfgen[20] plug in program within VMD1.8.1 or later version that is a very useful to create PSF file.

   (b) **topology top_all22_port.inp**

   Loads the topology file *top_all*22*_port.inp* in order to read the topology definitions for the residues of *BPTI*.

   (c) **segment BPTI pdb 6PTI_protein.pdb**

   Builds the segment of *BPTI* that contains all atoms (sequence of residues) of 6PTI_protein.pdb. Also it adds hydrogen atoms. Some error reading first and, last residues are normal[20].

   (d) **patch DISU BPTI:5 BPTI:55**

   (e) **patch DISU BPTI:14 BPTI:38**

(f) **patch DISU BPTI:30 BPTI:51**

Add some patch residues after the segment $BPTI$ is built. These contain all angle and dihedral terms explicitly. The patches are applied for the disulfide link [20].

(g) **alias residue HIS HSE**

Changes the residue name of histidine to the proper name found in the topology file. HSE is one of three names for histidine, based on the protonation state of its side group. [See appendix for Histidine residues].

(h) **alias atom ILE CD! CD**

The atom named $CD1(\alpha carbon)$ in the isoleucine residue is renamed as "CD", its proper name from the topology file. Since isoleucine contains only α carbon atom. The PSF file dose not use the number label "CD".

(i) **coordpdb 6PTI_protein.pdb BPTI**

Coordinates are read from 6PTI_protein.pdb, residues and atom names are matched. Old segment labels are overrides by $BPTI$.

(j) **guesscoord**

Coordinates of missing atoms (like hydrogen) are guessed based on residue definitions from the topology file. These guessed atoms will be positioned to the correct places in the minimization step.

(k) **writepdb BPTI.pdb**

A new PDB file with complete coordinates of all atoms including hydrogen, is written.

(l) **writepsf BPTI.psf**

A PSF file with complete structural information of the protein is written.

3. Force field parameter file of CHARMM is available in the internet. At this point the PSF and the PDB files of $BPTI$ are ready for a simulation in the vacuum environment, but for a solution environment there is a need to solvate and ionize the $BPTI$ in a water box.

2.1.2 neutralized of $BPTI$

The $BPTI$ protein needs to be solvated, i.e., put inside water (solution environment) so as to be similar to the cellular environment.

There are several reasons for solvating the $BPTI$. Many biological processes occur in aqueous solution. Solvation effects play a vital role in determining molecular conformation electronic properties, binding energies, etc.

Solvation of *BPTI* is done in two steps. First, the solvent molecules (water molecule) are added to the molecular system *BPTI*. Second, the solvent is modeled as a continuum dielectric environment (water environment).

To build the *PSF* and the *PDB* files for *BPTI* in water box, the following commands are written at *VMD* prompt:

- **package require solvate**

  Places the solvate package in the correct place. *VMD* will be now able to call it for adding water molecule.

- **solvate** *BPTI.psf BPTI.pdb −t* **6** *−o* **BPTI_water**

  The solvate package will put *BPTI* described in BPTI.psf, BPTI.pdb (generated before) in a box of water that is large enough to avoid the interaction between the protein and its image (from the other side) in the next cell of Periodic Boundary Condition (PBC) (see appendix C.3). The -t option used to create the water box dimension such that there exist a layer of water 6 $\AA$ (Angstrom) in each direction. The -o option creates the output files BPTI_water.psf and

BPTI_water.pdb for *BPTI* with water box. The water box should be large enough, this mean the water will still significantly immerse the protein when it is fully extended in order to be sure protein is inside the water box[49].

- **set everyone [atomselect top all]**
- **measure minmax** $*everyone*

  Analyzes all atoms in the system and gives the minimum and maximum values of x, y, and z coordinates of the entire protein water system. These *minmax* values are defined relative to the origin of the coordinates system *BPTI* that are set by the initial *pdb* file. The center of the water box is determined by calculating the midpoint of each of the three box sides x, y, and z coordinates (next command line).

- **measure center** $*everyone*

  Determines the center coordinates of the water box center.

2.1.3 neutralization of *BPTI* in a water box

Ions are placed in the water to represent a typical biological environment. The ions are especially necessary if the protein carries an excess charge. In that case, the number of ions should be chosen to make the system neutral. The ions present will protect the regions which carry charge, and make the entire protein environment more stable. They should be placed in regions of potential minima because they will be forced to those regions during the simulation. The *PSF* file contains the charge of each atom, which is used to determine the net charge of the molecular system (protein).

```
proc get_total_charge molid top {
eval "vecadd [[atomselect molid all] get charge]"
}
```

Returns the total charge of molecular system where molid is an molecular index passed to the procedure that determine which protein to chose.

```
package require autoionize
```

Running autoionize with no arguments gives a short overview of the syntax. You can add ions in either of the following two ways: Specify the number of ions, or let the package calculate the net charge and neutralize the protein.

```
autoionize −psf BPTI_water.psf −pdb BPTI_water.pdb −is 0.05
```

Tells autoionize to compute the sodium and chlorine ion numbers so that the net charge of the system is zero, and the average ionic strength of the solution is (in this case) 0.05. Alternatively, autoionize place the given numbers of the sodium (NNa) and chlorine (NCl) ions as user want. In this case, the system may not be electrically neutral. autoionize provide additional options:

$$\begin{cases} -o < prefix >, & \text{output file prefix (default 'ionized');} \\ -from < distance >, & \text{min distance from molecule (default 5A);} \\ -between < distance >, & \text{min distance between ions (default 5A).} \end{cases}$$

2.1.4 Simulation of BPTI in both Vacuum and Solution

NAMD Configuration File

A NAMD configuration file contains a set of dynamics options and values that control how the molecular system will be simulated. The number of timesteps to perform, initial temperature, etc are some of these options. Each line in the configuration files consists of an case insensitive keyword and a value. The keyword and the value are separated only by a space or equal sign and space(Appendix C).

Any $NAMD$ configuration parameter file must has these necessary parameters: numsteps (number of steps), coordinates, structure, parameters, exclude, outputname, and one of the following: temperature, velocities, or binvelocities (binary velocity format). The previous parameters specify the most basic properties of the molecular dynamics simulation. The

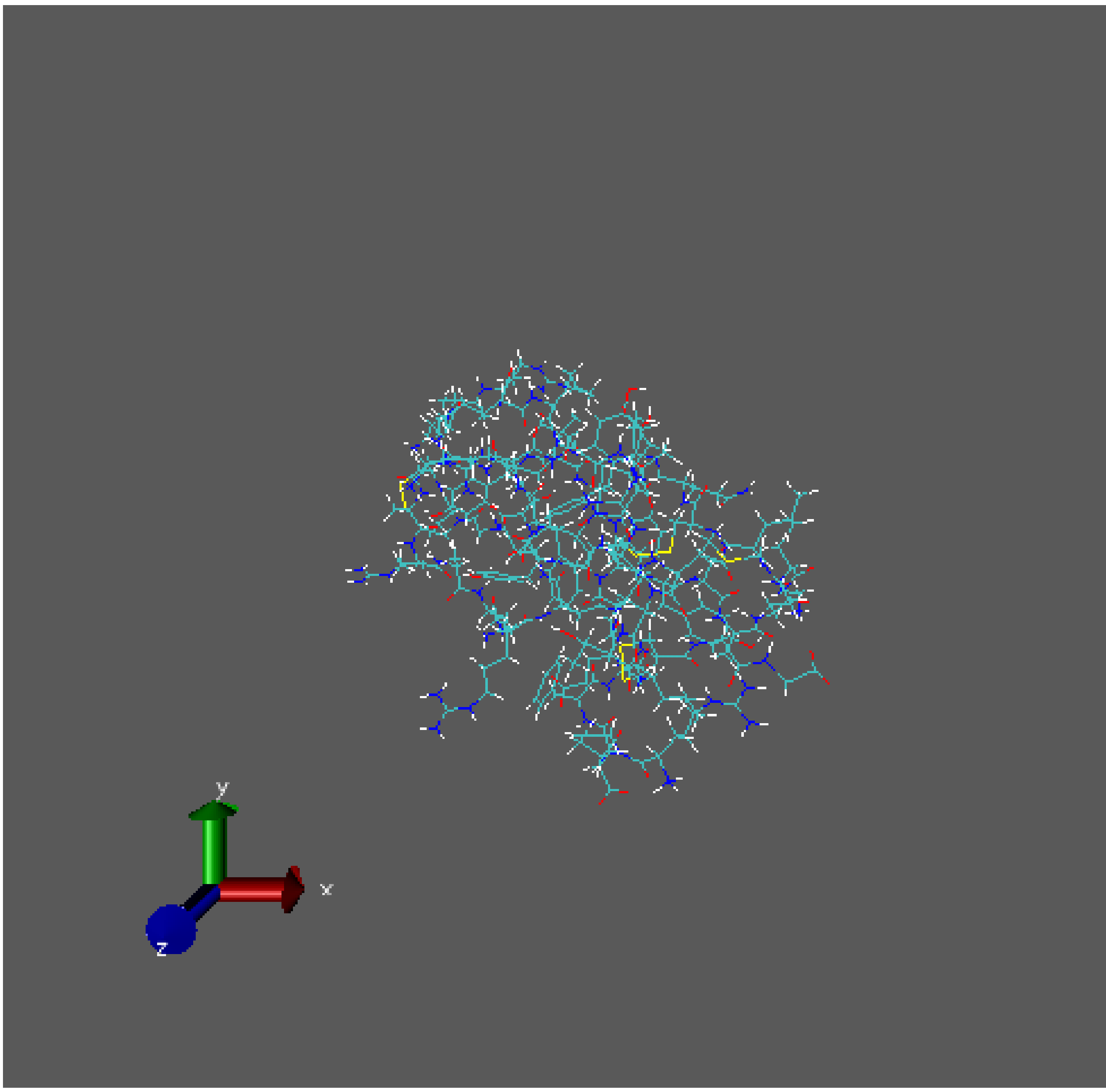


Figure 2.2 BPTI in vacuum

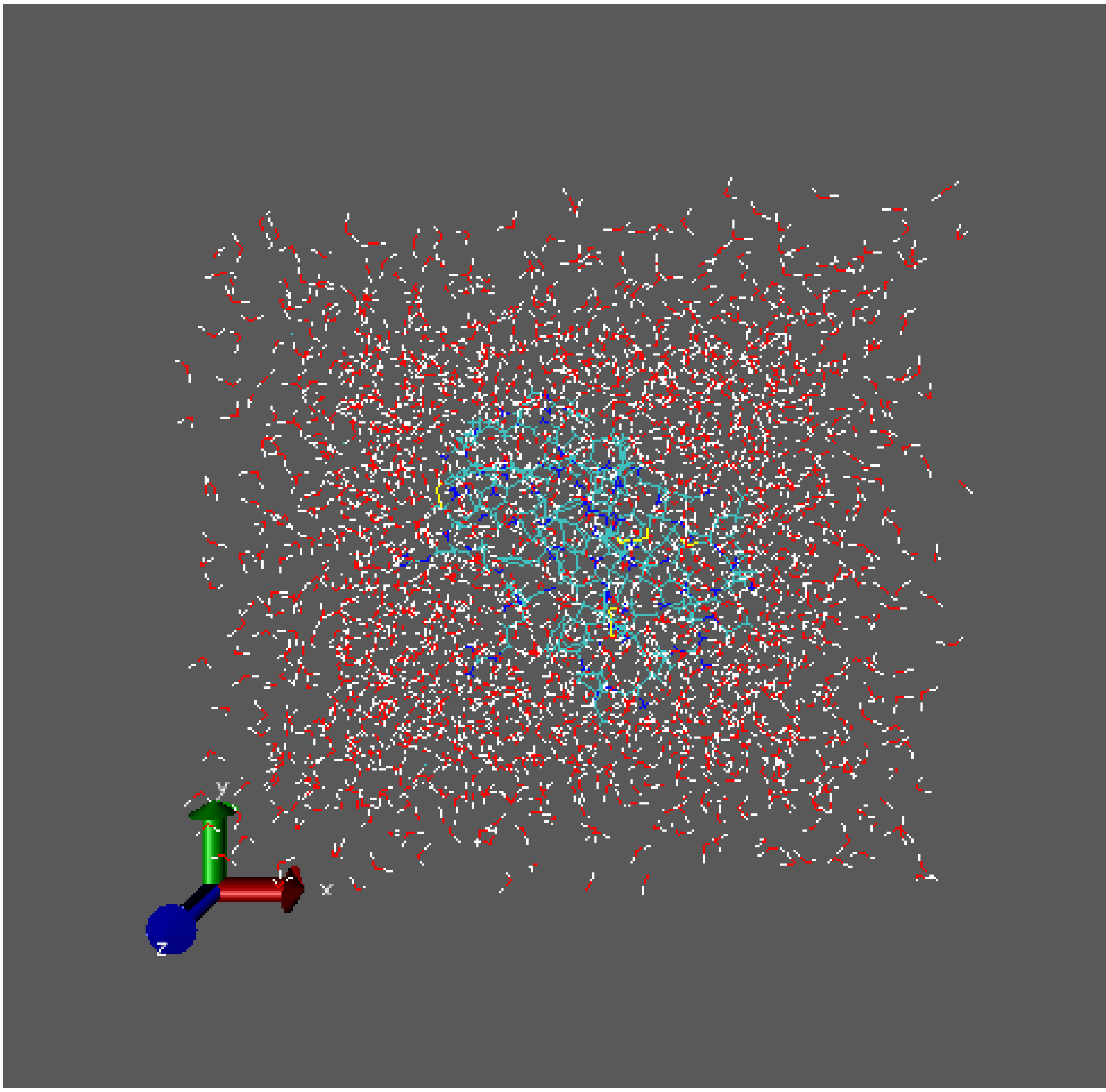


Figure 2.3 BPTI in Solution after ionization

configuration parameters files for $NAMD$ in both environment is given in Appendix C.

2.1.5 Minimization

The $BPTI$ system was minimized for 500 steps with the Conjugate Gradient method using $NAMD$ in order to remove any strong Van Der Waals interactions that may exist. Otherwise these lead to local structural distortion and result in an unstable simulation[13].

2.1.6 Heat

Initial velocities at zero (K) temperature are assigned (now each atom have zero velocity)to each atom of the system and newton's equations of motion are integrated to propagate the $BPTI$ in time. The $BPTI$ was heated from zero to 300 K in 20 ps (picoseconds) time scale using langevin dynamics method.

2.1.7 Equilibration

Once the desired temperature of 300 K is reached, the vacuum simulation of $BPTI$ continues and during this phase several properties are monitored. In particular, the pressure, the temperature, and the energy on the system[13] as well as the structure. The aim of the equilibration phase is to run the simulation until these properties become stable with respect to time. If the temperature increases or decreases significantly, the velocities can be scaled such that the temperature returns to near its desired value. The $BPTI$ was equilibrated for 40 ps in 300 K using the temperature reassignment parameters method[20].

2.1.8 Dynamics: Control production phase

The final step of the simulation is to run the simulation in production phase for 20ns. The production of the 20 ns MD simulation was done at 300 K. Appendix C shows the control parameters and values for the vacuum and the solvent simulations such as timestep, space partitions, basic dynamics, electrostatic, multiple timestep algorithm, constrains, fixed atoms, temperature control, and the periodic boundary condition[20].

The coordinates of all atoms were saved for analysis every 50 steps in trajectory DCD files (Appendix B.6), giving a total of 400000 structures (frames) of total size (30) GB (GigaByte) for solution and 400000 structure of size (5) GB for vacuum.

2.1.9 Analysis

The total trajectory DCD files will contain all information about all atoms' positions in the protein during the course of the simulation. The $C\alpha$ atoms in the protein contain the interesting information[51, 1, 32] of the protein states. The tool *catdcd* [45]was used to filter out (separate) these atoms $C\alpha$ from the trajectory DCD files. First, the file containing the index for $C\alpha$ is prepared using the VMD tool. After opening the protein in VMD the following lines are typed at the prompt

- >> **set ca [atomselect top "name CA"]**

  >> **$ca get index**

  >> **$ca writepdb ca.pdb**

The index is copied and saved in a text file called index.txt. The last line is used to save only $C\alpha$ atom information in PDB format for later use. (Note in VMD program CA means $C\alpha$ alpha carbon)

The instruction that allowed to separate $C\alpha$ atoms from DCD trajectory files is: (*Note* Go to the directory where *catdcd* installed (bin *catdcd* directory) and type the following at windows or unix prompt)

$ ./catdcd -i index.txt -o calpha.dcd /root/simulationresult/yourTrajectoryfile.dcd

This line produce a new trajectory DCD file called calpha.dcd in the bin directory of *catdcd* containing only C-alpha atoms trajectory. To deal with the big sized DCD files, it is helpful to store the $C\alpha$ trajectories in several files. *Catdcd* is a powerful tool for splitting and combining DCD files. Load ca.pdb that was created before using VMD with the new DCD

file "calpha.dcd".

There is a need to fit and align the protein structures (frames) to a reference frame. This frame is chosen as the first frame after the simulation has fully equilibrated at the desired temperature. In this work the structure after 250ps was chosen as the reference frame.
The following *tcl*[44] script procedure (in *VMD* editor) fit and align the protein structures:

- **proc fit_align_structure {{mol top}} {**
- **# use frame 5000 for the reference**
- **set reference [atomselect $mol "name CA" frame 5000]**
- **$reference writepdb ref.pdb**
- **# the frame being compared**
- **set compare [atomselect $mol "name CA"]**
- **set num_steps [molinfo $mol get numframes]**
- **for {set frame 5000} {$frame $num_steps** {incr frame} {
- **# get the correct frame**

- **$compare frame $frame**
- **#compute the transformation using Kub algorithem**
- **set trans_mat[measure fit $compare $reference]**
- **# do the alignment**
- **$compare move $trans_mat**
- **} #end of loop**
- **} #end of script**

After that go to the *VMD* Display window to check if the protein is fitted and aligned to the reference structure (frame), then save the new coordinates in a new *DCD* file. This file contains the fitted and the aligned trajectory. Note when *VMD* saves the coordinates it assumes a unix platform even if the operation is performed from a windows platform. Therefore, The *DCD* file is written as binary little endian. So you must know which platform and which byte order the *DCD* file has. This is easily done using *flipdcd* tool that comes with the *NAMD* distribution package.

Go to the *NAMD* bin directory and type the following at windows or unix prompt

$./flipdcd yourDCDfile.dcd

This command Gives the byte order and converts it from little endian to big or vice versa. We can now use the *VMD* prompt to calculate the RMSD values for all structureS after the fit and align was done by typing the following

- **proc get_rmsd {{mol top}} {**
- **set f [open vacuum_rmsdall.dat a]**
- **# use frame 5000 for the reference**
- **set reference [atomselect $mol ”name CA” frame 5000]**
- **# the frame being compared**
- **set compare [atomselect $mol ”name CA”]**
- **set num_steps [molinfo $mol get numframes]**
- **for {set frame 5000} {$frame $num_steps} {incr frame} {**
- **# get the correct frame**
- **$compare frame $frame**
- **# compute the RMSD**

- **set rmsd [measure rmsd $compare $reference]**
- **puts $f "$rmsd"**
- **}**
- **close $f**
- **}**

Plot the RMSD values you get in the file called *vacuum_rmsd_all.dat* as shown in Fig2.7.

The structures are now read from the *DCD* file and loaded to a java classes' anadcd.java (Appendix A.2) in order to find the covariance as follows:

1. This class "anadcd.java"(Appendix A.2) finds the covariance matrix (equation 1.64)and the "B-factor" (equation 2.1) (Temperature Factor) of the protein)[52].

2. Plot the "B-factor" (equation 2.1, Appendix A.2) to compare it with the experiment values from protein data bank. Temperature-factor" or "Debye-Waller factor (Fig 2.4) is a factor that can be applied to the X-ray scattering term for each atom (or for groups of atoms) that describes the degree to which the electron density is spread

out. While the theory is that the B-factor indicates the true static or dynamic mobility of an atom, it can also indicate where there are errors in model building[23]. B-factor calculated from course of MD simulation as in (equation1.61)

$$B_i = \frac{8}{3}\pi^2 < \Delta r_i^2 >= \frac{8}{3}\pi^2 < c_{ii} > \tag{2.1}$$

$$\Delta r_i = r_i - < r_i >_t \tag{2.2}$$

where Δ is the mean-square fluctuation of atom i, c_{ii} is covariance (equation 1.64), and r_i is trajectory position of particle i.

3. Use correlation.java class (Appendix A.2) to find the correlation map matrix.

$$correlation_{ij} = \frac{c_{ij}}{\sqrt{c_{ii} * c_{jj}}} \tag{2.3}$$

4. Carma tool was used to plot the correlation map (equation 2.3) by typing the following in carma binary directory under unix prompt

   $cat corrolationmap.txt $\|./carma\_gcc -$

This produced a figure in ps format at the same directory of carma bin called stdio.carma.ps. Rename this figure to overcome the problem of reproduce other picture of the same name as shown in (Fig2.9).

5. After that Matlab was used to find the associated eigenvector and eigenvalue for the covariance matrix.

6. Plot the eigenvalue of each $C\alpha$ with respect to its index in matlab as seen in Fig2.15.

7. Use class eigproject.java (Appendix A.3)class to make a projection with each eigenvector for each structure. For example, the dot product between the first eigenvector values and trajectory data structure (frame), repeat the dot product again with all structures this is also must done with each eigenvectors. For simplicity do the projection with the significant eigenvectors and two or three nonsignificance eigenvectors.

8. Plot the projection values for each eigenvector with respect to structure index as shown in Fig2.18 and Fig2.23.

9. Plot the distribution of the projection values as shown in Fig2.31.

10. Plot the projection values on planes defined by projections between two eigenvectors as shown in Fig2.25.

11. Form a correlation map by taking the dot product between eigenvectors from different covariance matrices and plotting them using the carma tool as shown in Fig2.36.

2.2 Analysis of results

2.2.1 B-Factors

To validate simulation results, different properties are usually calculated and compared to experimental data. These properties include structural properties such as crystallographic temperature factors (B-factors), energetic properties such as heat capacity, and dynamical properties such as viscosity[53].

In this work, B-factor experimental data, readily available from X-ray diffraction data for almost any protein from protein data banks [19], were used to check the validity of simulation results. The B-factor value for the atom is proportional to the mean-square fluctuation for the position of that atom. X-ray diffraction is used to determine atom electron densities. These densities are smeared due to the motion of atoms. If a Gaussian model is used to fit the spread of the electron density around the average position of each atom, the width of the Gaussian can be related to the "Debye-Waller factor"(equation 2.4), or B-factor (equation 1.61), by the following equation [54]:

$$(\Delta r_i)^2 = \frac{3B_i}{8\pi^2} = c_{ii} \tag{2.4}$$

where Δ is the mean-square fluctuation of atom i, c_{ii} is covariance (equation 1.64), and r_i is trajectory position of particle i.

Figure 2.4 shows B-factor values calculated for water simulations ranging in time from 1ns to 20 ns, compared to experimental (crystallographic) values. The following observations were made:

1. Except for two regions, the first between the 10^{th} and 20^{th} residues, and the second between the 36^{th} and the 40^{th} residues, the simulation results give good qualitative agreement with the experimental values.
2. The simulation results do not show strong dependence on the time length of each simulation. This suggests that the B-factor values converge quickly.
3. The B-factor values in the two regions mentioned above increase considerably with time.

Figure 2.5 shows B-factor values calculated for vacuum simulations ranging in time from 1ns to 20 ns, compared to experimental (crystallographic) values. The following observations were made:

1. The simulation values give good qualitative agreement with the experimental values, except at the 48^{th} and 49^{th} residues.
2. The simulation values are consistently less than the experimental values, again except at the 48^{th} and 49^{th} residues.

3. The simulation values show almost no dependence on the length of the simulation.

B-factor values calculated from simulations usually give only qualitative agreement with crystallographic values [40] as shown in Hunenberger work (Fig2.6). However, it is standard procedure in molecular dynamics simulations to perform this test to check for the validity of the simulation before further analysis is performed.

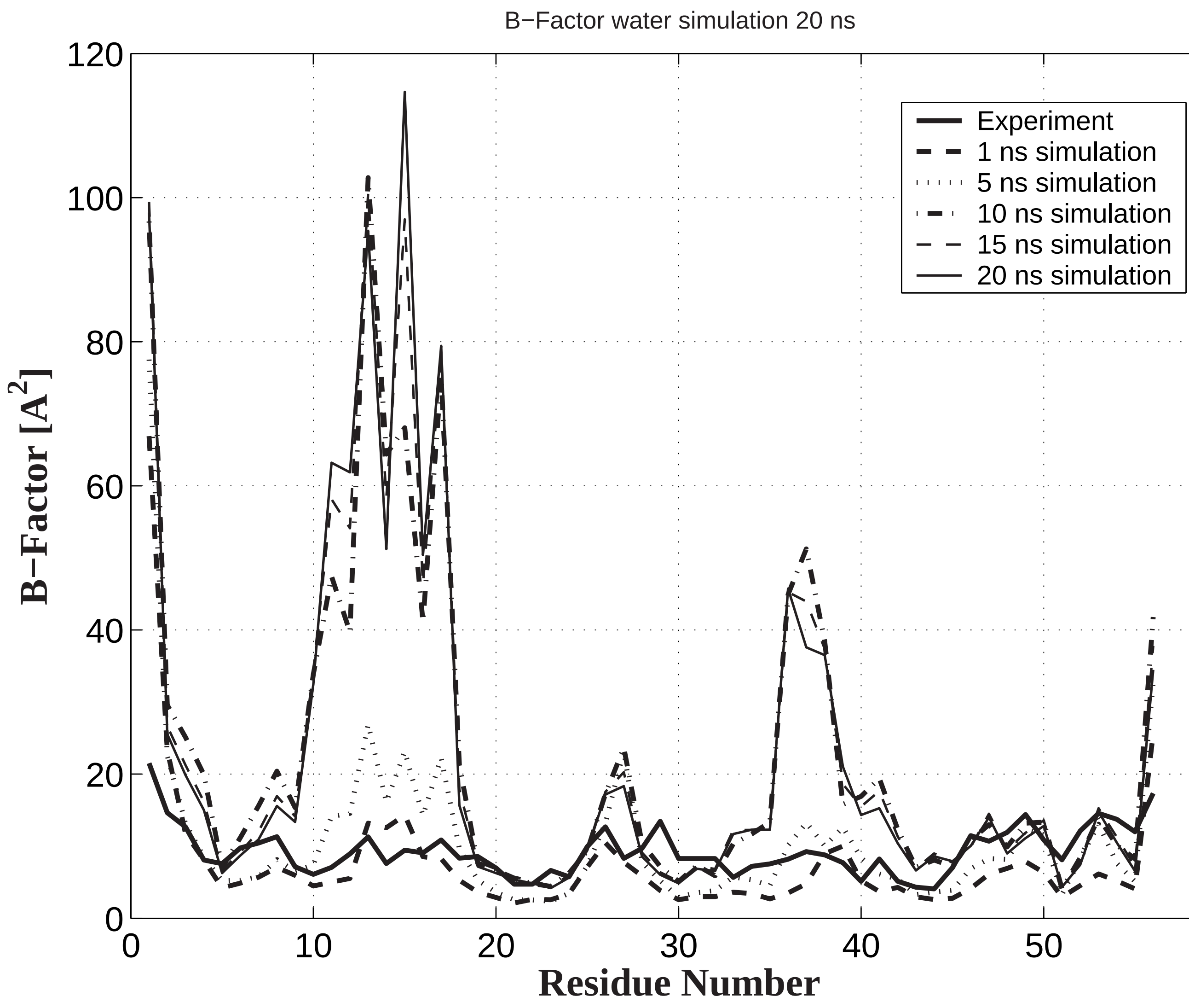


Figure 2.4 Comparison of rms fluctuations from simulation and from crystallographic B-factors. Fluctuations computed for $C\alpha$ (57 atom) atoms sampled from 0-20 ns water simulation.

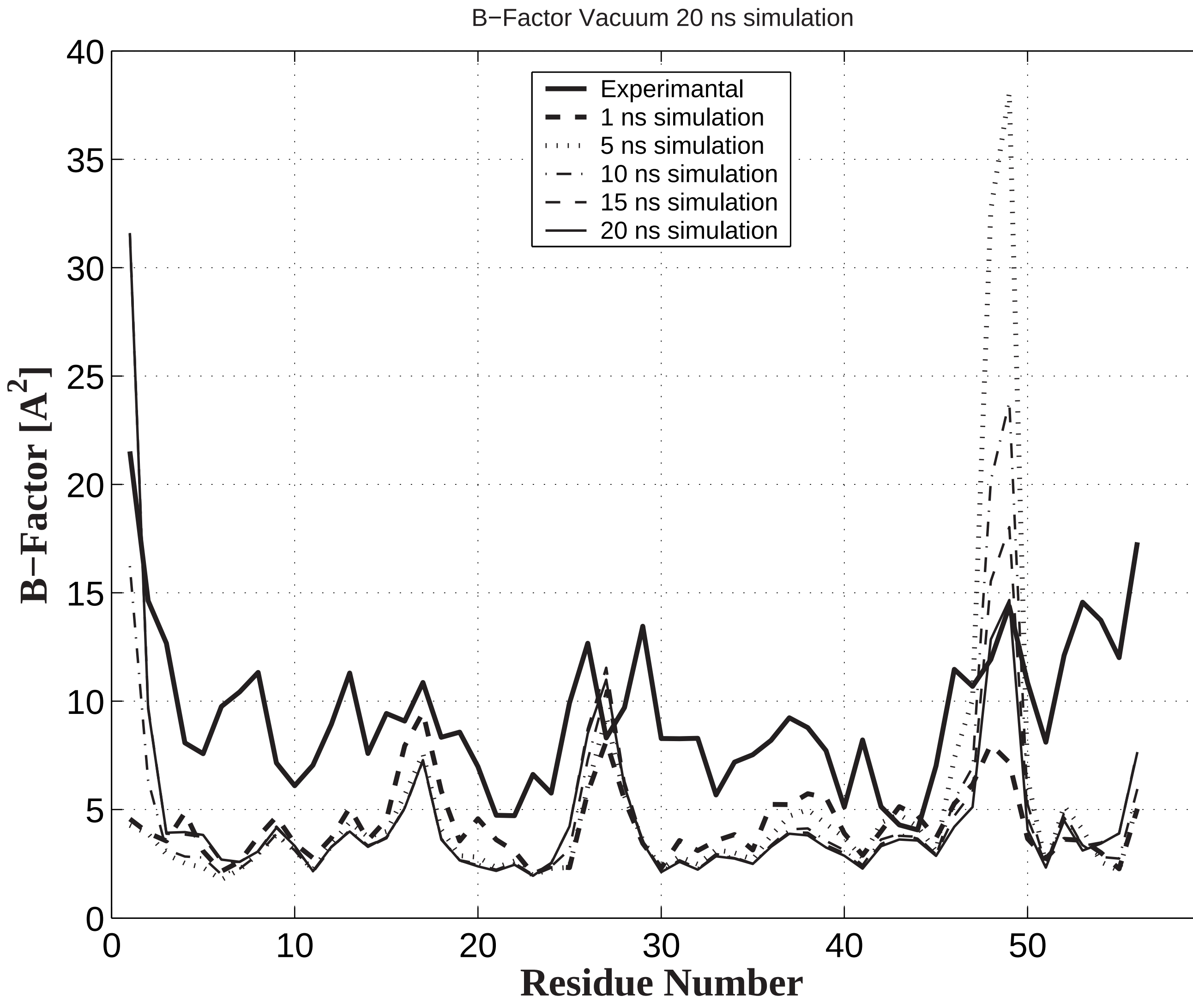


Figure 2.5 Comparison of rms fluctuations from simulation and from crystallographic B-factors. Fluctuations computed for $C\alpha$ atoms (57 atom) sampled from 0-20 ns vacuum simulation.

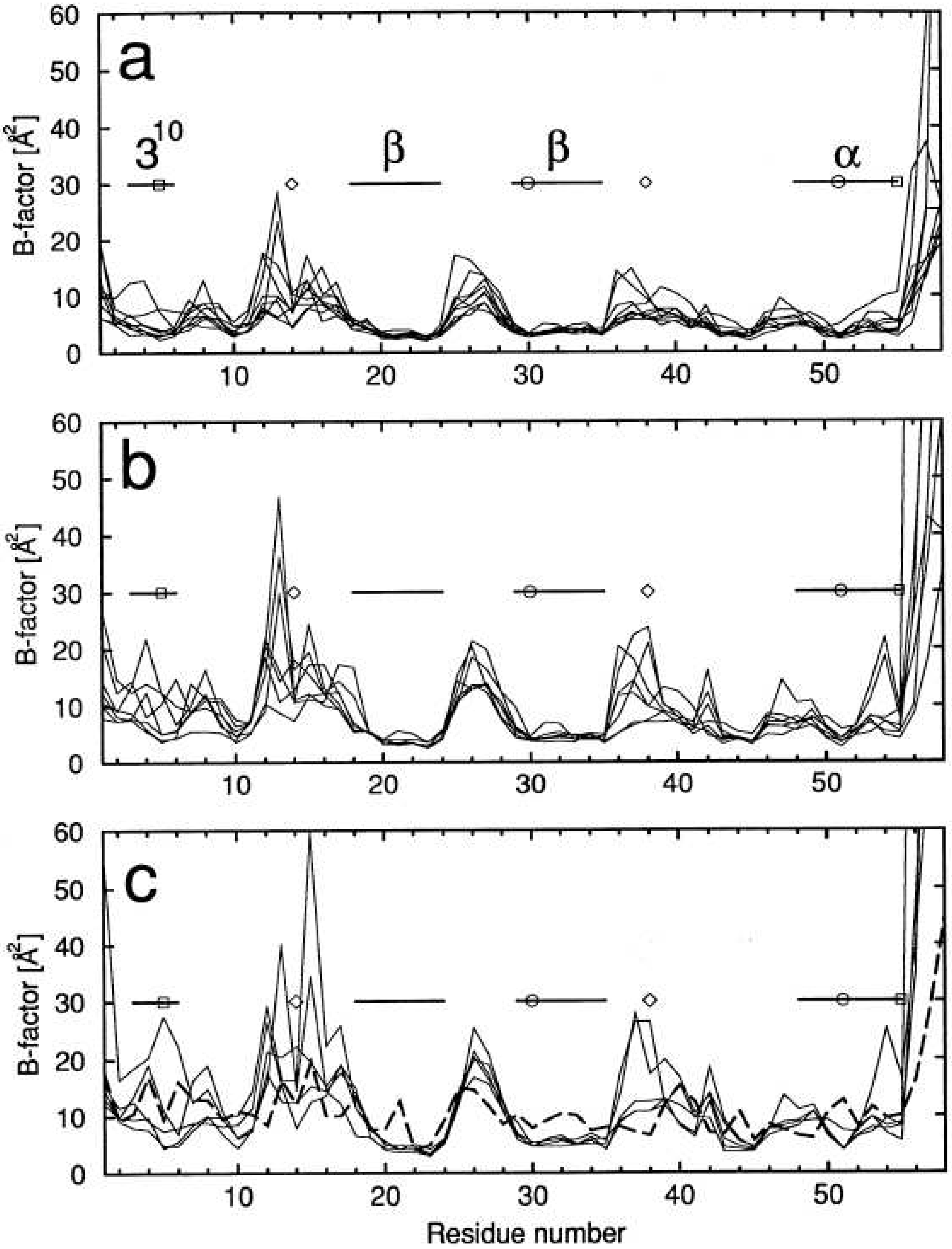
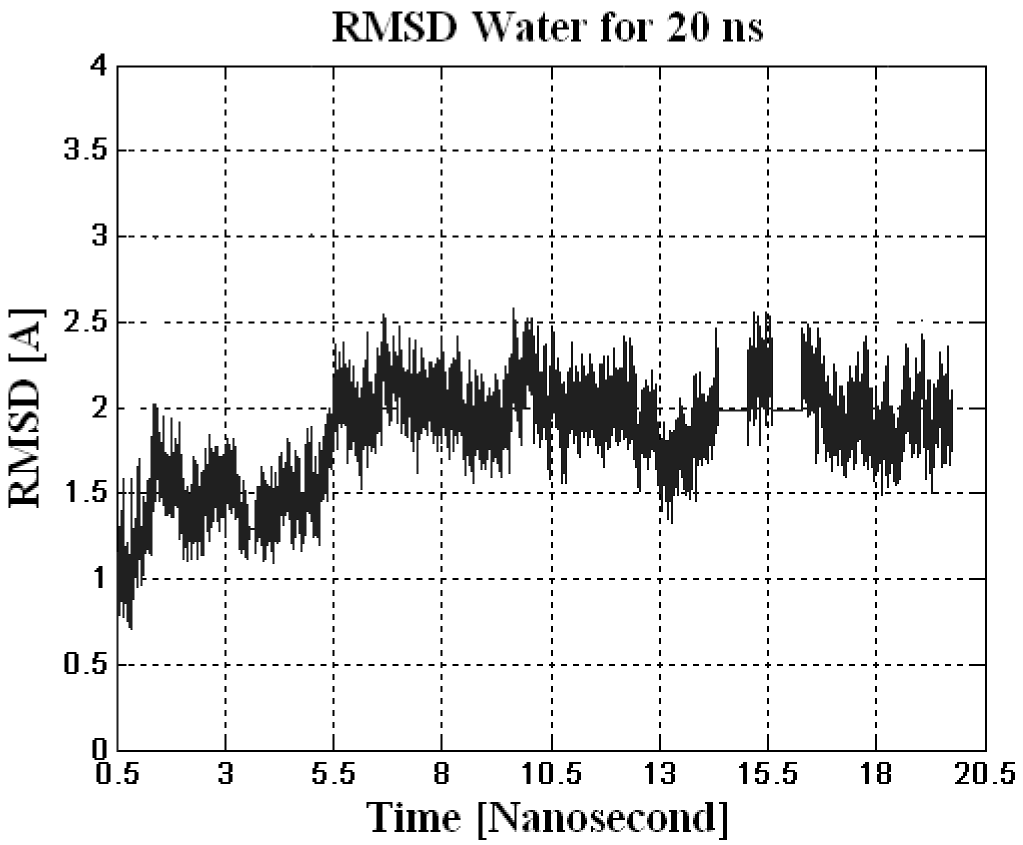


Figure 2.6 The Hunenberger [40] crystallographic B-factors for different window length.

2.2.2 Root Mean Square Deviation RMSD:

The calculated root mean square deviation RMSD of the calculated molecular dynamics structures from the initial equilibrated structure is shown versus time for the water simulation (Fig2.7 top), and the vacuum simulation (Fig2.7bottom). The equilibrium value for the water simulation converges to an average value close to 2 angstroms. The equilibrium value for the vacuum simulation converges to a value of 1.45 angstroms. Hueneberger et.al. [40] reported an average value of 1.82 angstroms for a 1.1 ns water BPTI simulation. Similar values are reported for other proteins. This test is performed to evaluate the stability of the molecular dynamics simulation.



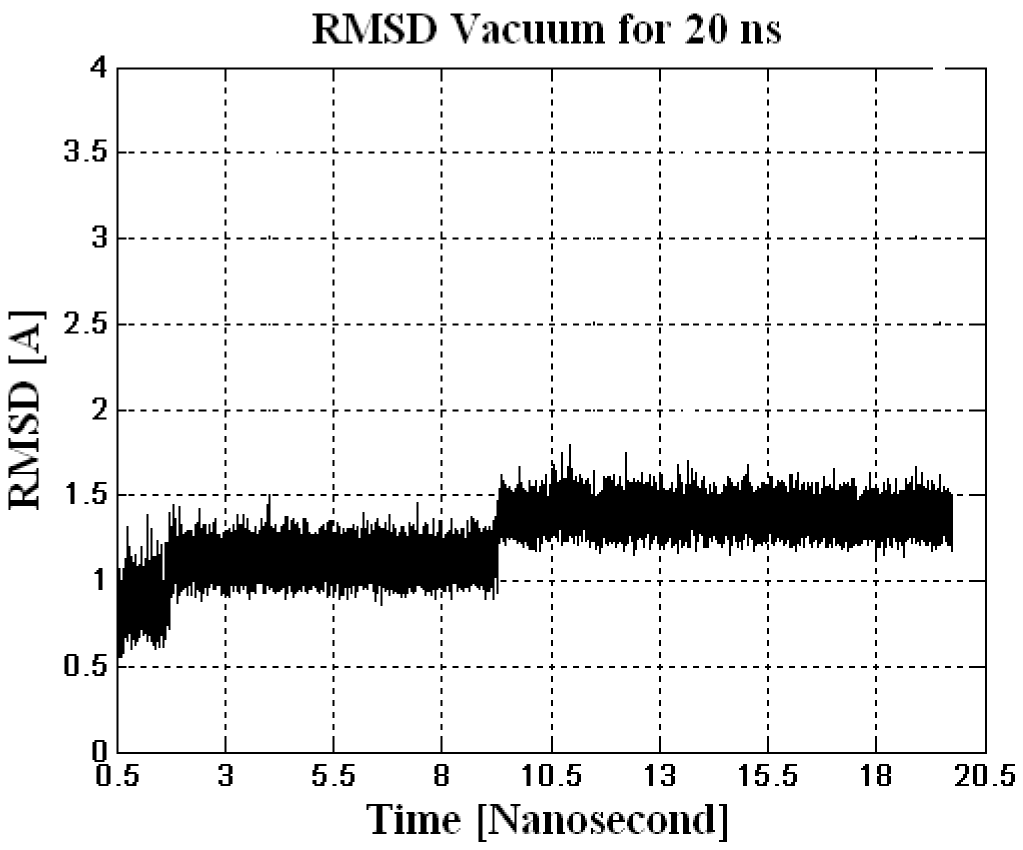
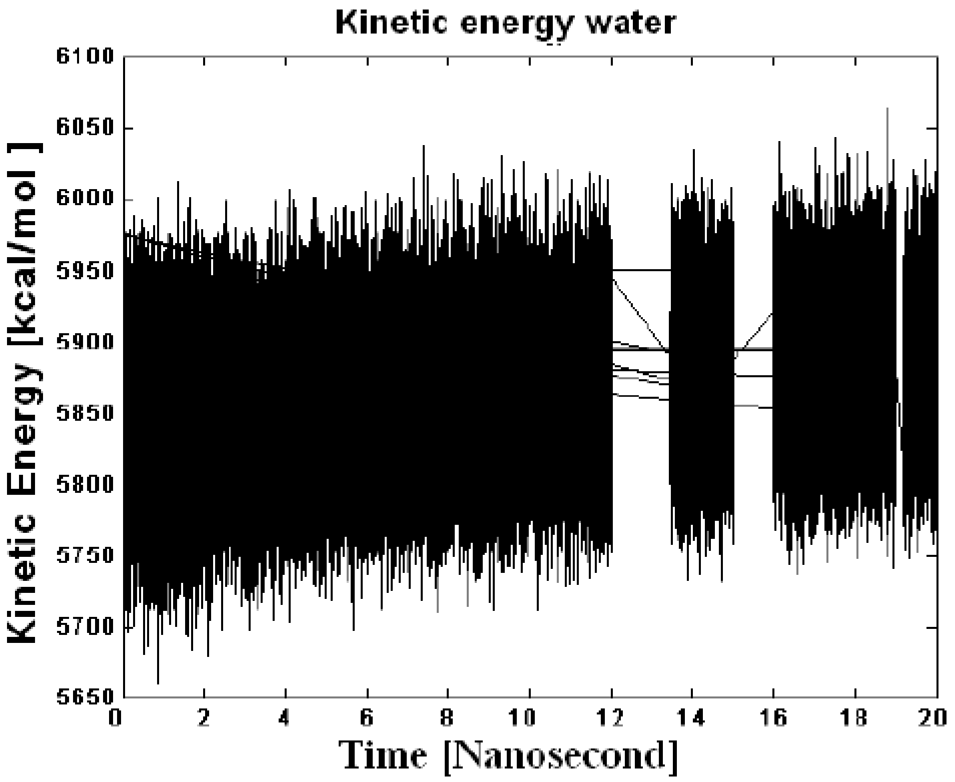


Figure 2.7 Calculated rms $C\alpha$ deviation (RMSD) of dynamic structures from the initial structure vs. time from the MD simulation for 20ns on $BPTI$.

2.2.3 Kinetic Energy:

The kinetic energy of the protein is directly proportional to its temperature. To check the stability of the simulation, the kinetic energy was calculated versus time. Simulations were performed at 300 K. Figure 2.8 shows how the kinetic energy of the protein behaves versus time. It converges in water to a value of 5850 [Kcal/mol] after 0.5 ns, and It converges in vacuum to a value of 680 [Kcal/mol] after 0.5 ns,. Thus the protein temperature remains constant throughout the simulation, a necessary condition for the validity of analysis results.



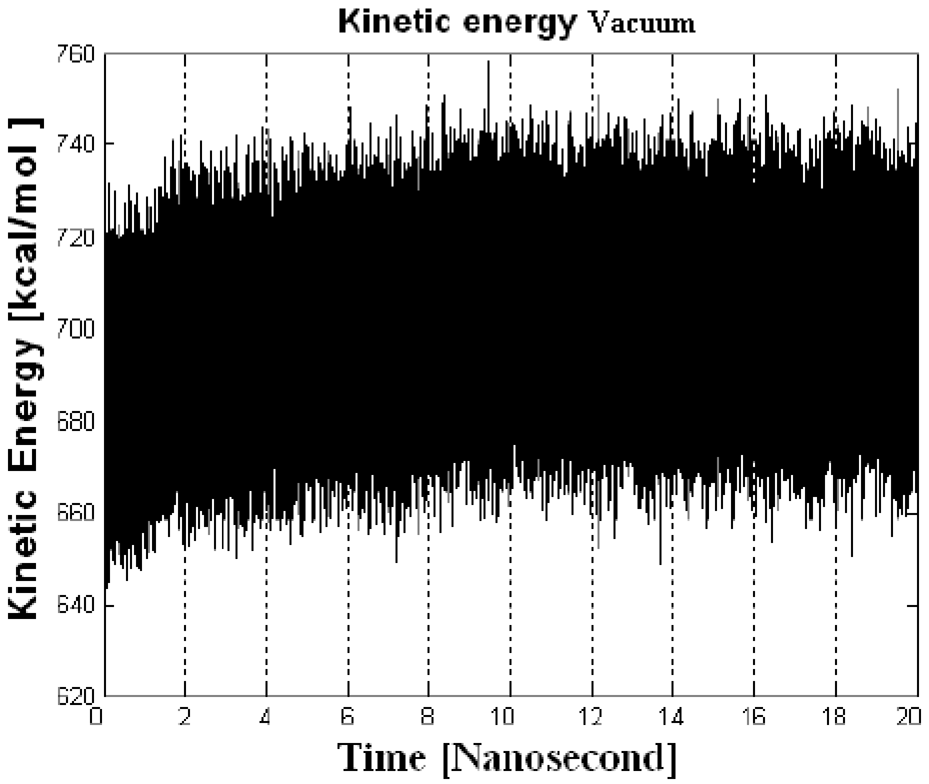
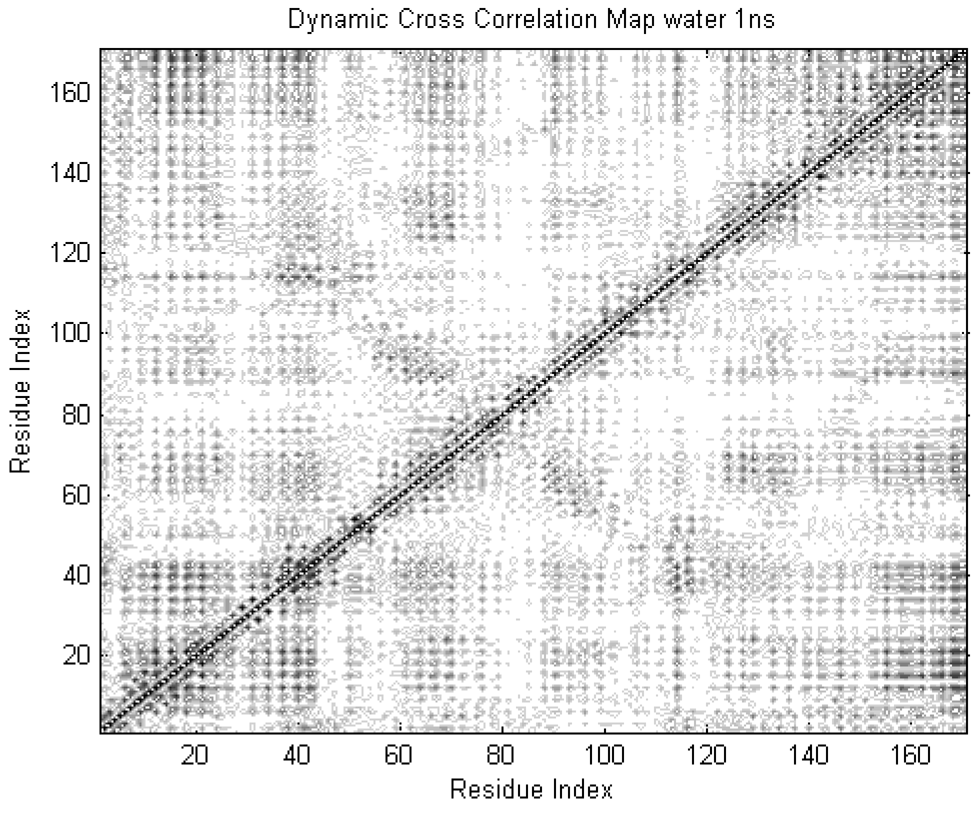


Figure 2.8 kinetic energy for 20ns simulation.

2.2.4 Correlation Maps:

Figures (2.9, 2.10, 2.11, 2.12, 2.13, 2.14) shows correlation maps of residues in the protein in both water (Top) and vacuum (Bottom). Each graph relates how the motion of each residue correlates with all other residues at a certain time. If both residues have large positive or large negative displacements from their average positions, then their correlation is close to 1, and this is shown as a dark spot on the map. If there is no correlation between the residues, the coefficient is close to zero, and is shown as a white spot. Negative correlation values where one residue has a large positive displacement, and the other has a large negative value, were also normalized to positive values, and are also shown as a dark spot with a value close to one. As is clear from the Figure 2.9, the water correlation maps show more dark areas signifying more correlation between residues. The correlation maps for vacuum show pretty much a constant picture of correlation very unlike the water simulation. These correlation maps might be the key to understanding which areas in the protein work together to perform different functions.



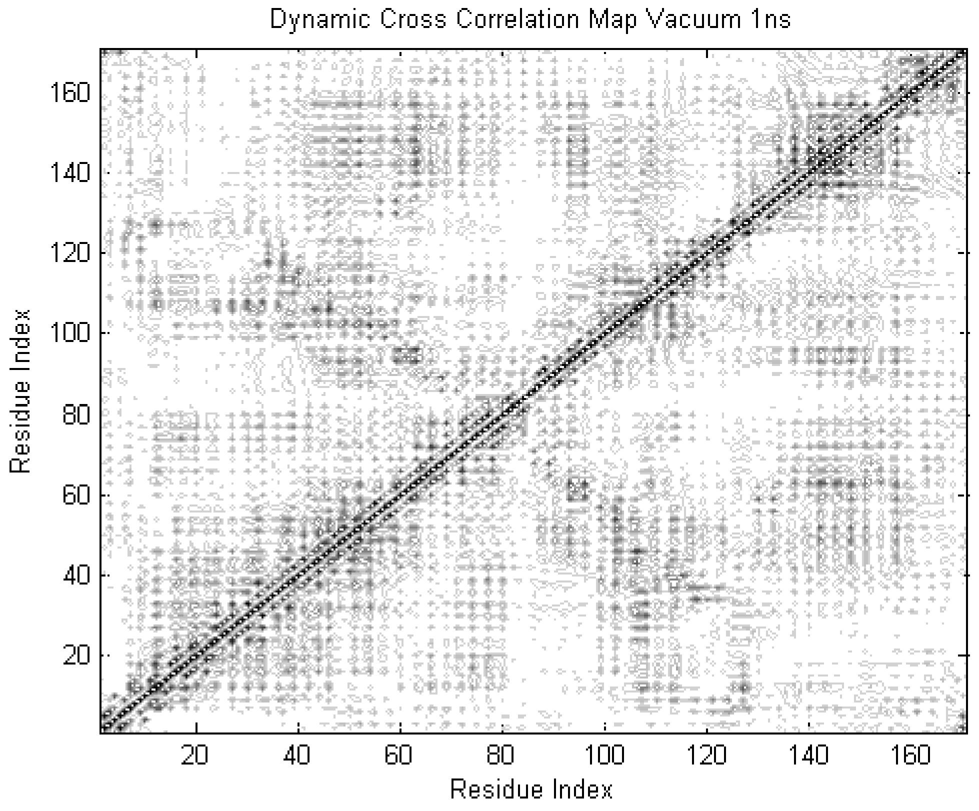
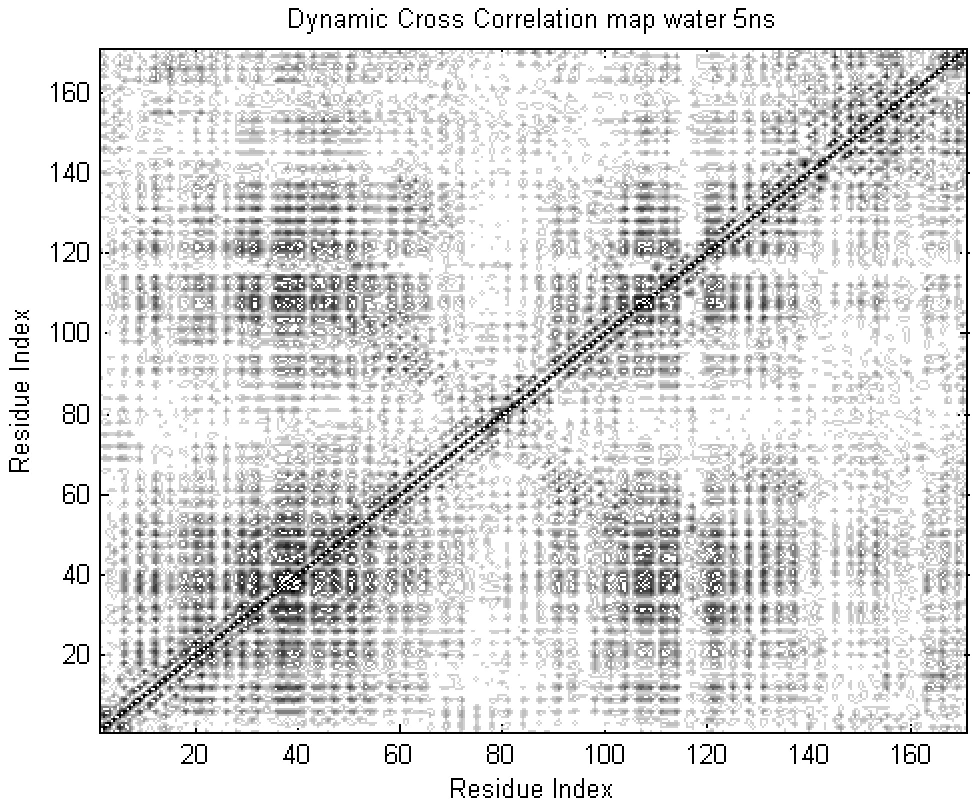


Figure 2.9 Calculated residue-residue-based correlated motions (Dynamic Cross Correlation Maps DCCM) after 1ns simulation for water (Top) and vacuum (Bottom)



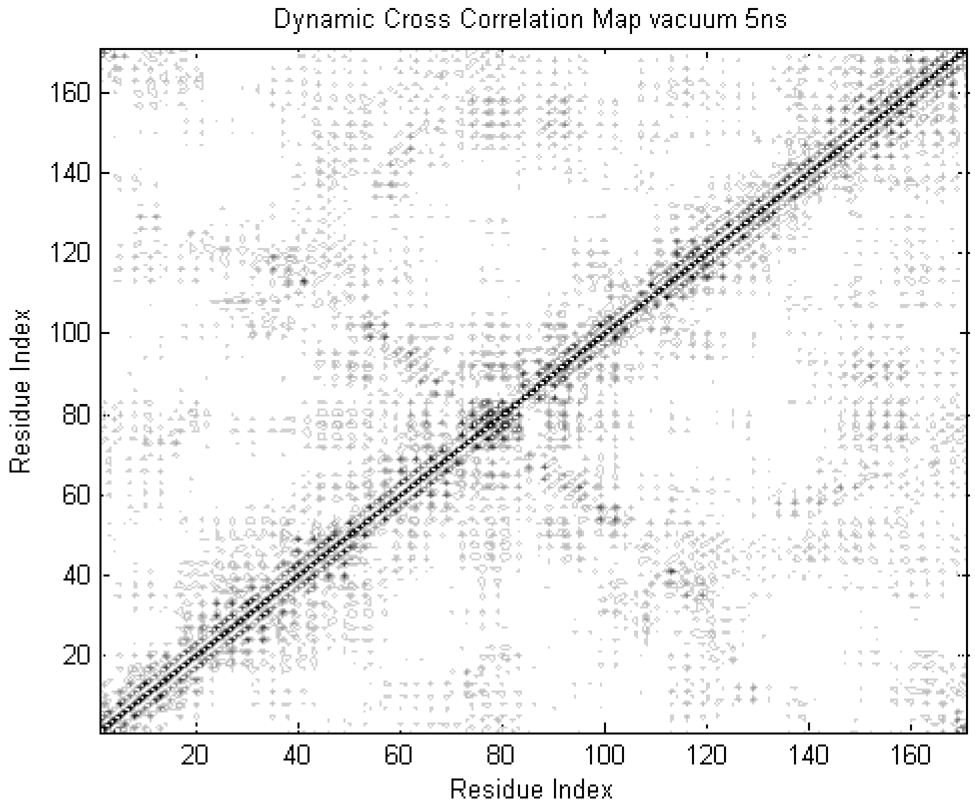
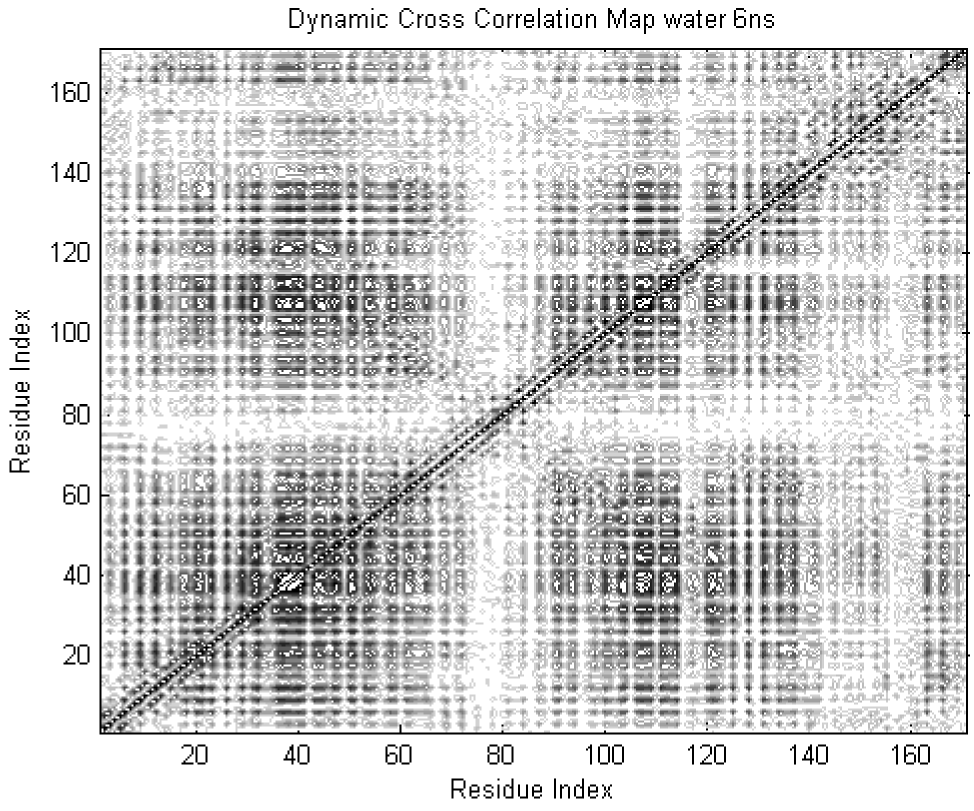


Figure 2.10 Calculated residue-residue-based correlated motions (Dynamic Cross Correlation Maps DCCM) after 5ns simulation for water (Top) and vacuum (Bottom)



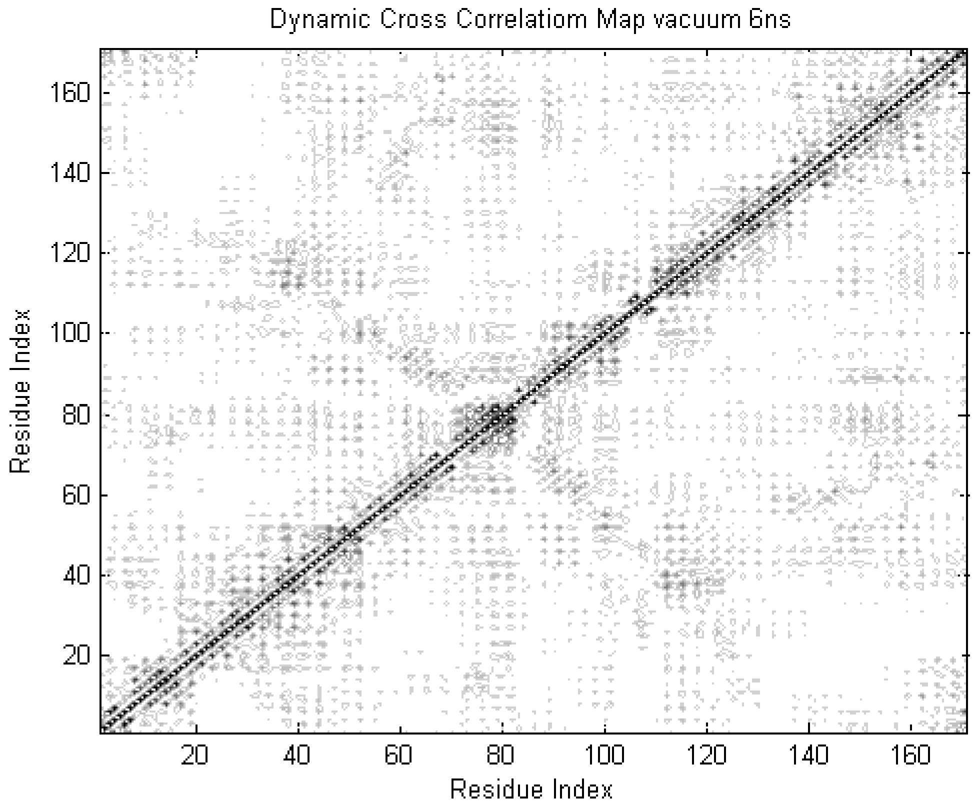
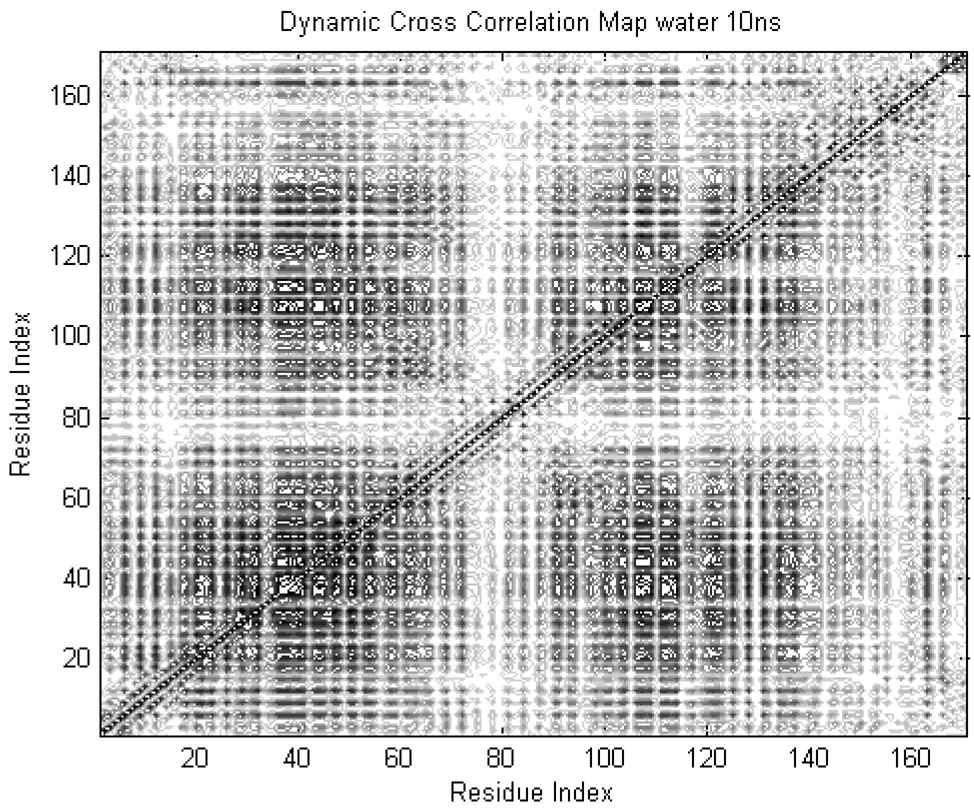


Figure 2.11 Calculated residue-residue-based correlated motions (Dynamic Cross Correlation Maps DCCM) after 6ns simulation for water (Top) and vacuum (Bottom)



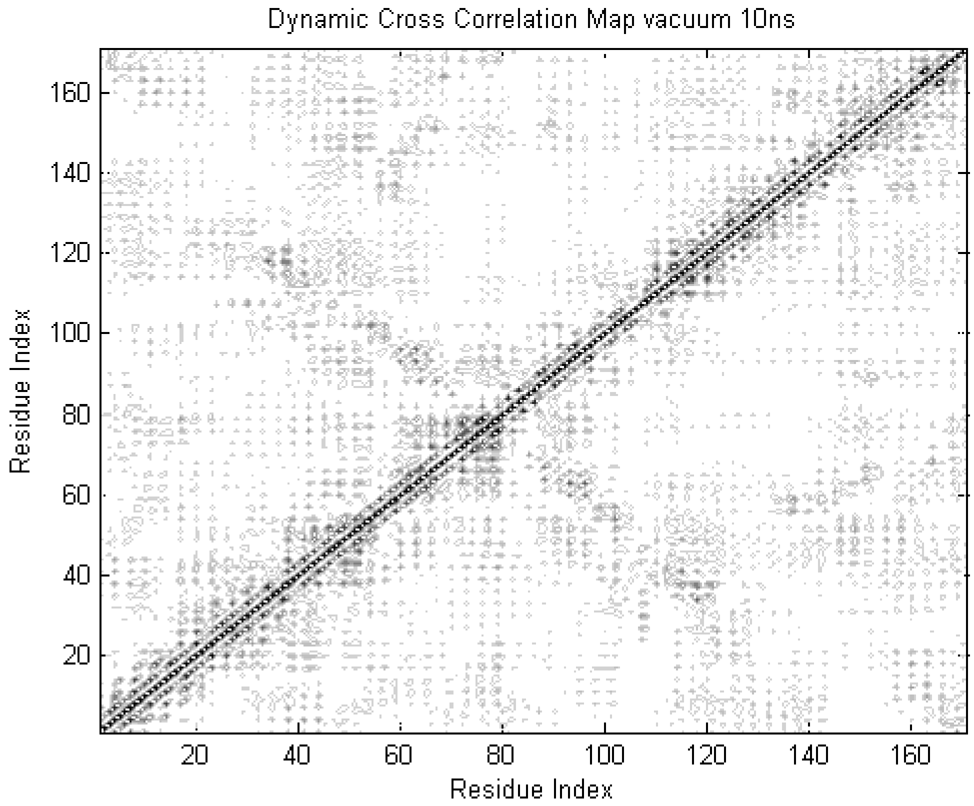
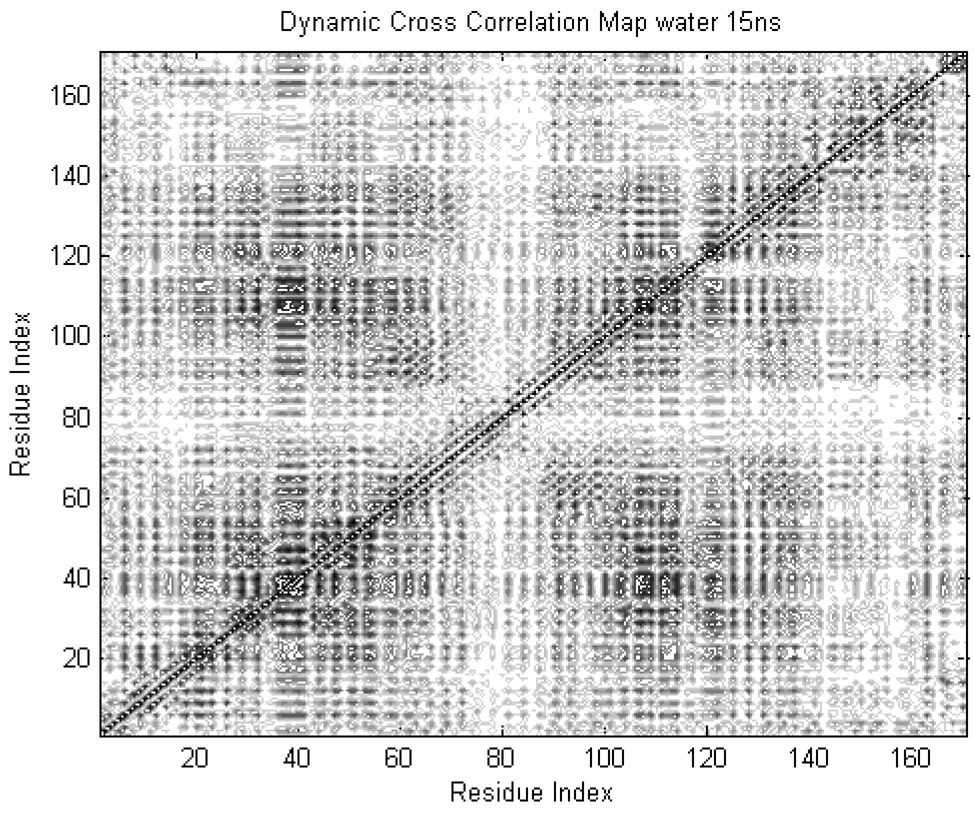


Figure 2.12 Calculated residue-residue-based correlated motions (Dynamic Cross Correlation Maps DCCM) after 10ns simulation for water (Top) and vacuum (Bottom)



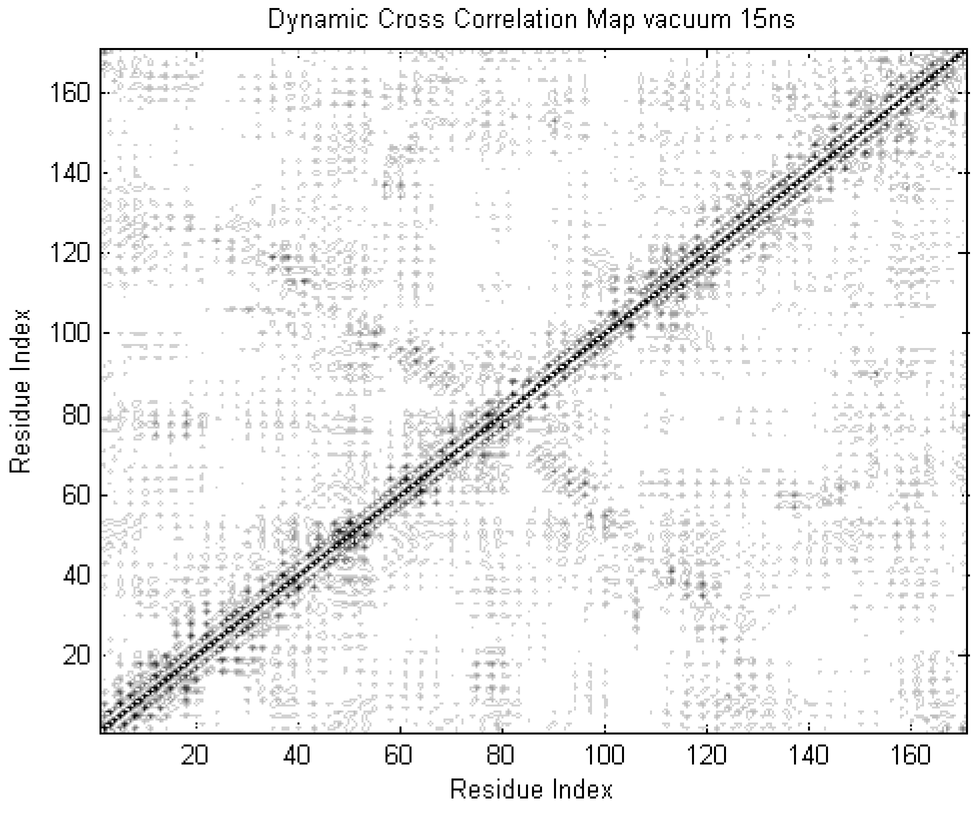
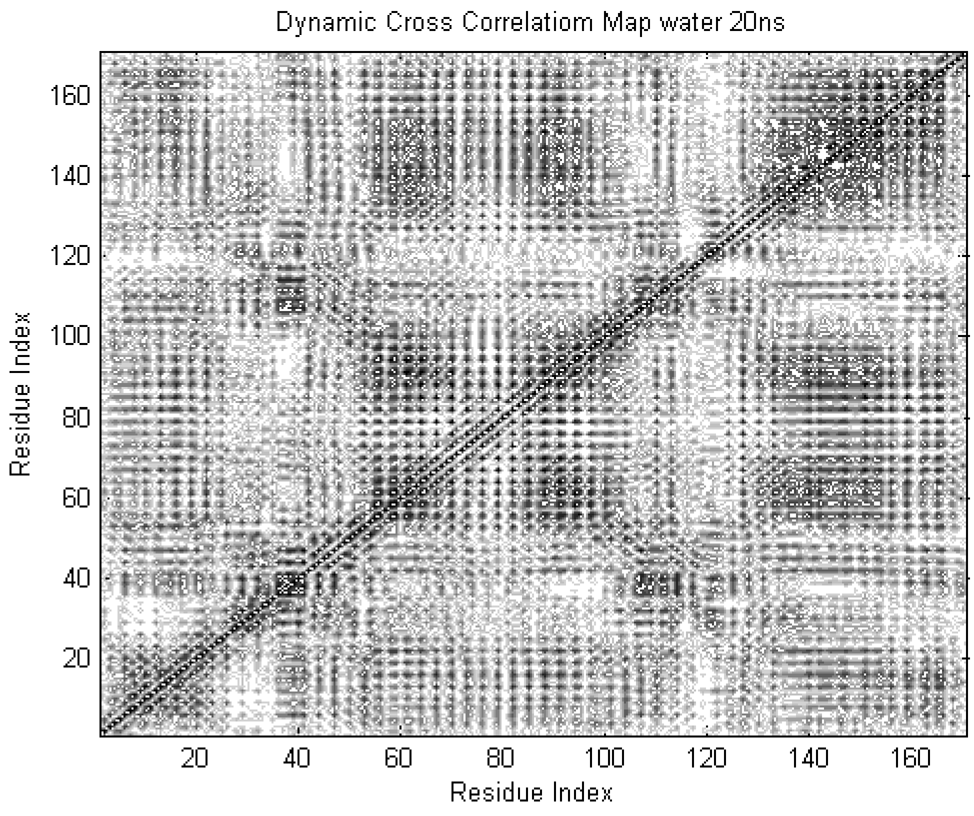


Figure 2.13 Calculated residue-residue-based correlated motions (Dynamic Cross Correlation Maps DCCM) after 15ns simulation for water (Top) and vacuum (Bottom)



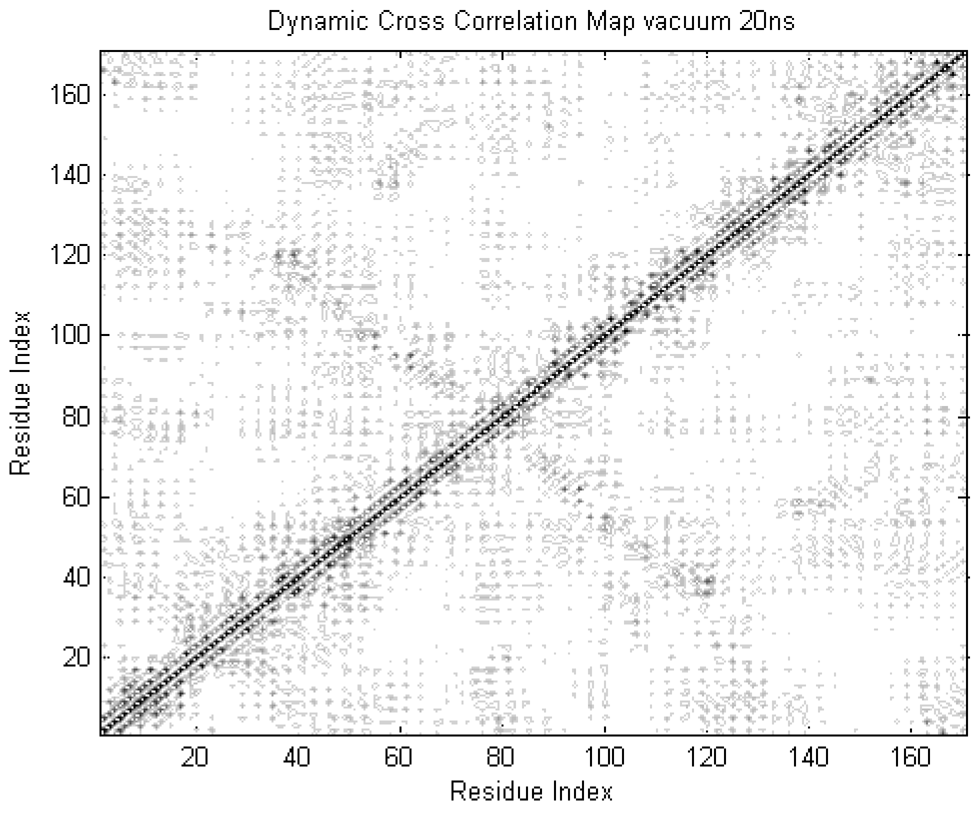
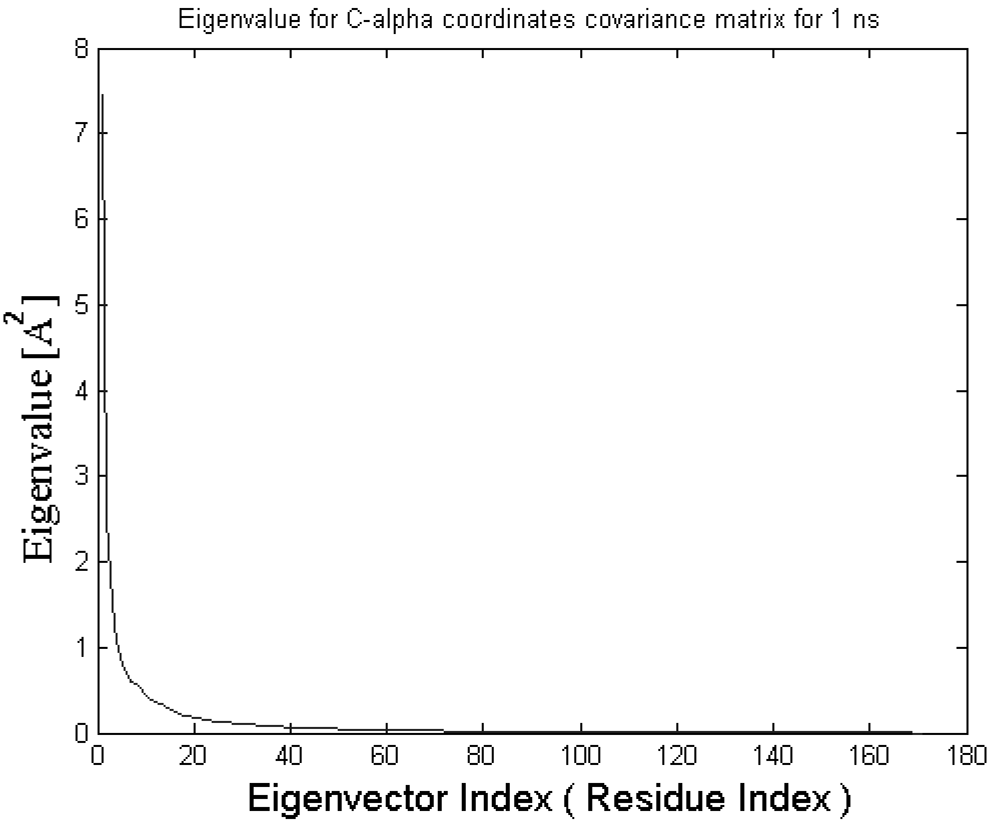


Figure 2.14 Calculated residue-residue-based correlated motions (Dynamic Cross Correlation Maps DCCM) after 20ns simulation for water (Top) and vacuum (Bottom)

2.2.5 Eigenvalues

Plots of the eigenvalues(amplitude of the atom motion), versus the eigenvector index calculated by diagonalizing the covariance matrix throughout the 20ns solvent simulation (Fig2.15 Top, Fig2.16 Top, and Fig2.17 Top) and the 20ns vacuum simulation (Fig2.15 Bottom, Fig2.16 Bottom, and Fig2.17 Bottom) are shown. The three figures at 1ns, 10ns, 20ns show a steep decrease in the eigenvalues after a few eigenvectors. This means that a few eigenvectors are enough to describe the protein motion, leading to a substantial simplification in the description of such a motion. The time window for each calculation was 500ps. An eigenvector with a large eigenvalue signifies a correlated motion among the atoms of the protein along that direction. This similarity in behavior, whereby the eigenvalues are significant for a few eigenvectors, suggests that short simulations might be sufficient to extract such information without the need for lengthy simulations.



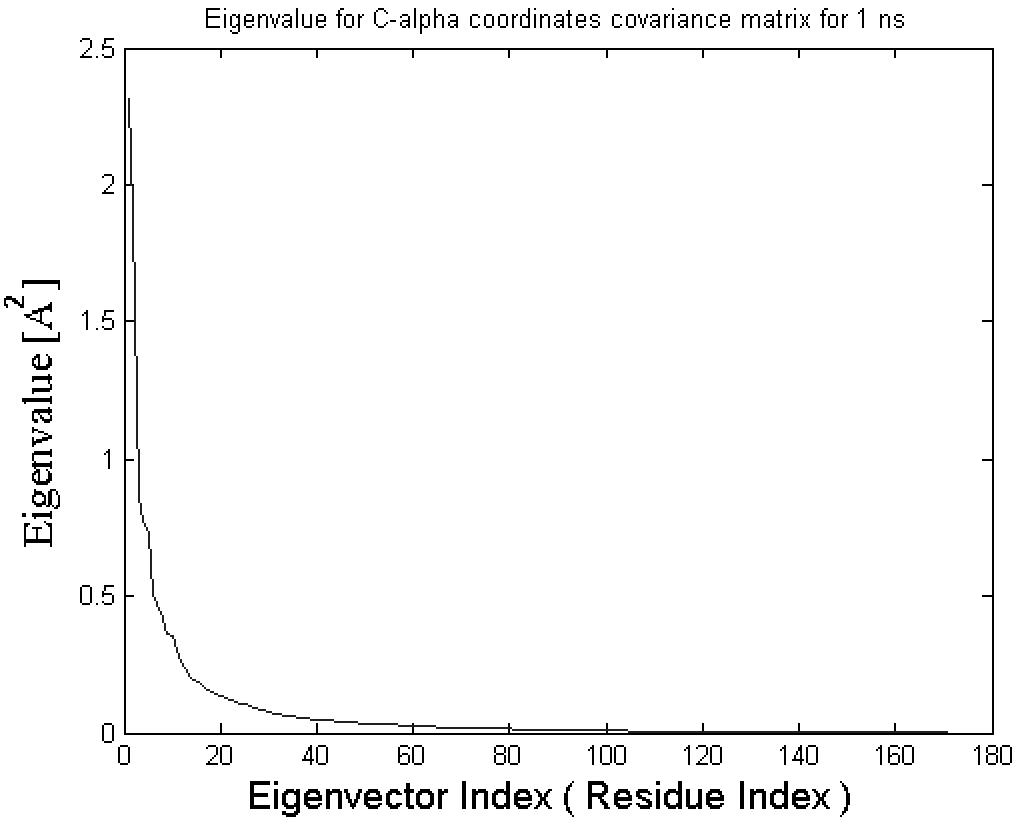
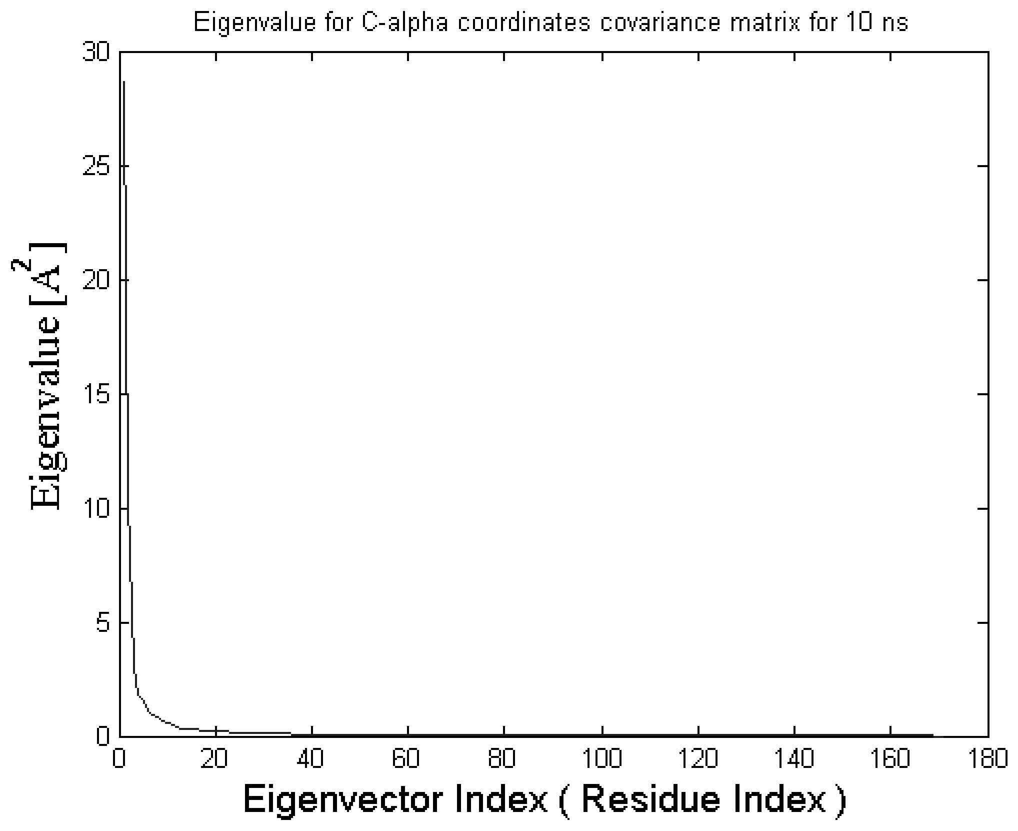


Figure 2.15 Eigenvalues, in decreasing order of magnitude for water (Top) and vacuum (Bottom) simulations obtained from $C\alpha$ coordinates covariance matrix after 1ns.



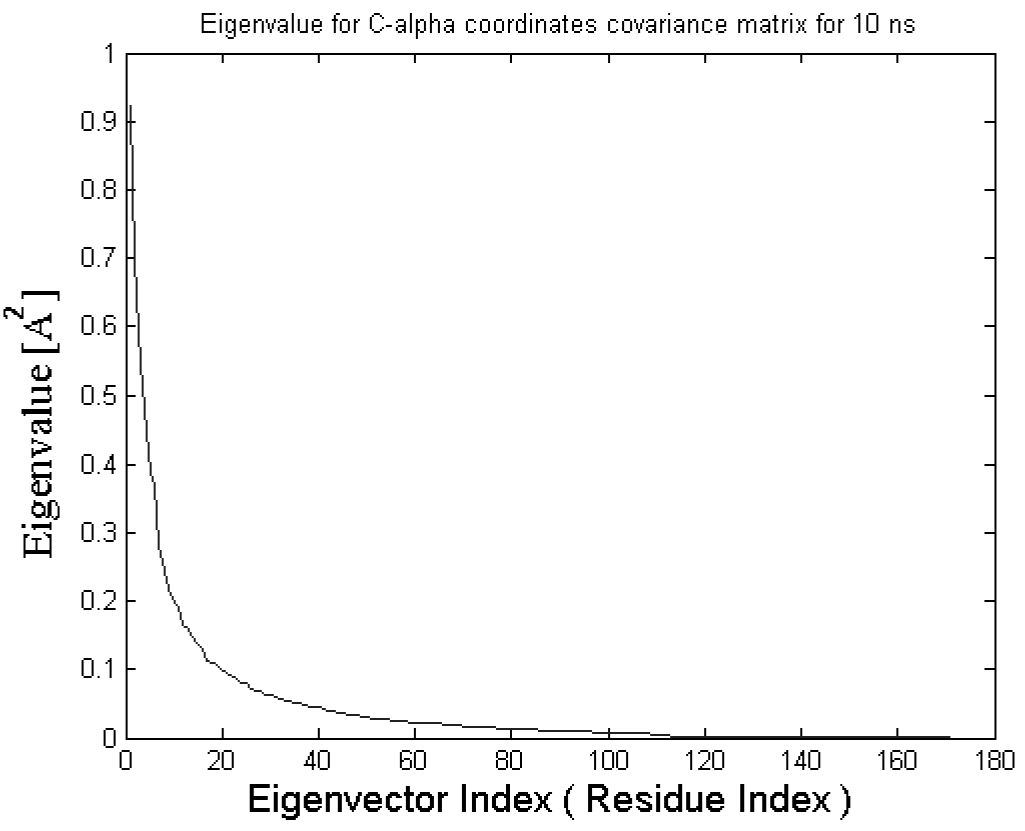
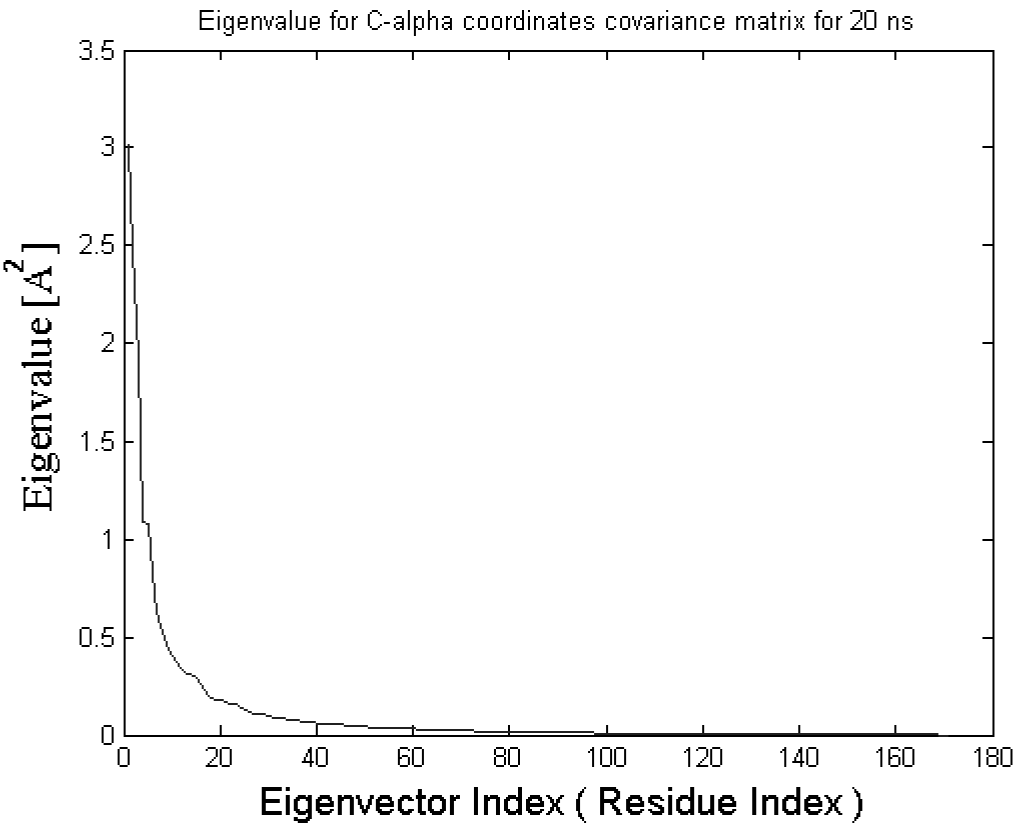


Figure 2.16 Eigenvalues, in decreasing order of magnitude for water (Top) and vacuum (Bottom) simulations obtained from $C\alpha$ coordinates covariance matrix after 10ns.



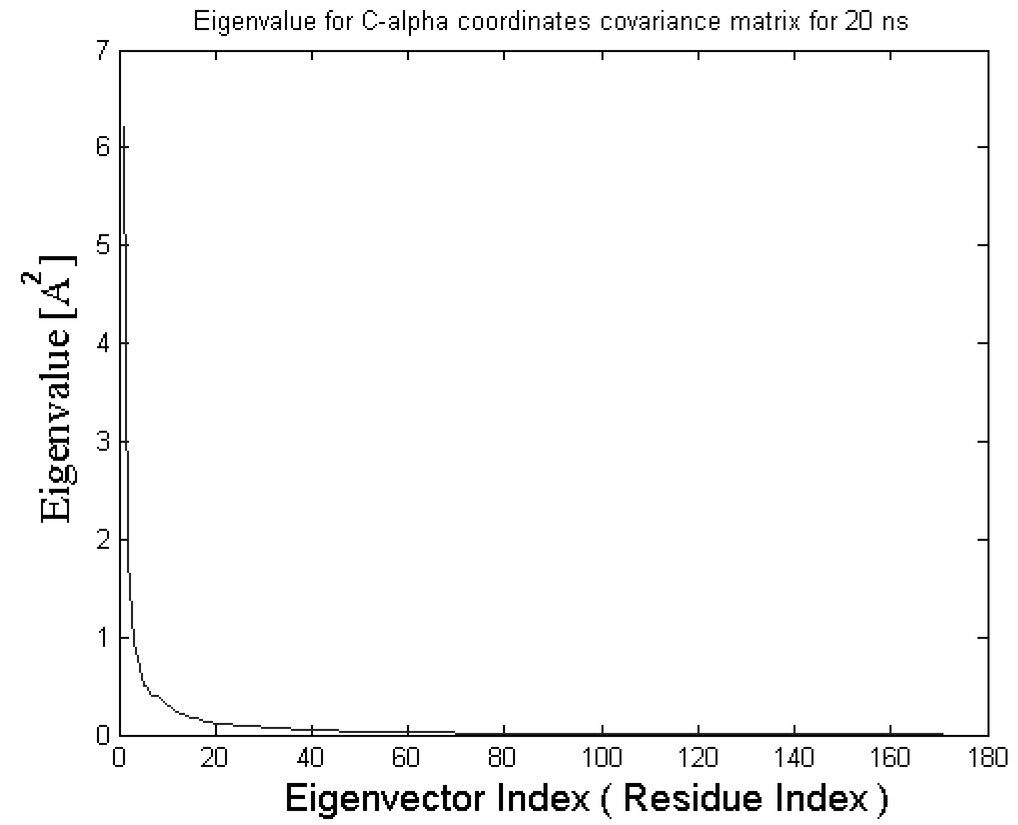


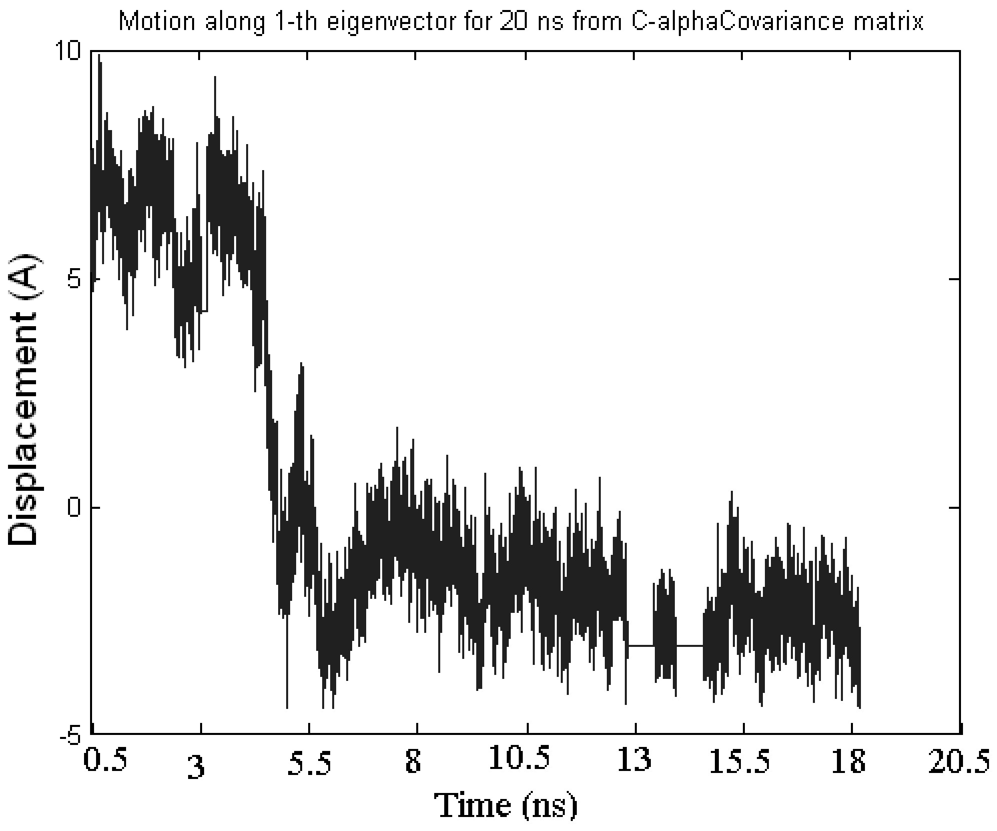
Figure 2.17 Eigenvalues, in decreasing order of magnitude for water (Top) and vacuum (Bottom) simulations obtained from $C\alpha$ coordinates covariance matrix after 20ns.

2.2.6 Motions Along Eigenvectors for 20 ns

It is possible to represent a trajectory at a certain instant in time by projecting it in the direction of one of the essential space eigenvectors. This is performed by taking the dot product between the atom coordinates of the protein at a certain time, and the components of an essential dynamics eigenvector. By repeating this procedure versus time, one can study the motion of the protein in a certain direction, and how it changes versus time.

The projection of the 20ns solution trajectory in the direction of five eigenvectors is shown in Top figures: 1^{st} (Fig2.18), 2^{nd} (Fig2.19), 3^{rd} (Fig2.20), 20^{th} (Fig2.21), 50^{th} (Fig2.22) for the solvent simulation (Top figures) and the vacuum simulation (Bottom figures). Each graph shows how the displacement of the protein trajectory changes with time along the eigenvectors. As can be seen in Top figures (Fig2.18), (Fig2.19), (Fig2.20), there is significant correlated motion along the first three eigenvectors. In figures Top (Fig2.21), (Fig2.22), the displacement basically hovers around zero. This indicates that the motions along these two directions are random in nature. The vacuum simulations in (Fig2.18), (Fig2.19), Fig2.20) Bottom) show no correlated motions, even along the directions of the first three significant eigenvectors. This was also true for the displacement along the 20th (Fig2.21 Bottom) and the 50th (Fig2.22 Bottom) eigenvectors. However, a sudden "jump" in the displacement is seen in Bottom of figures (Fig2.18, Fig2.21, Fig2.22 Bottom). It is not clear if this "jump" has significance, or is merely an artifact of the simulation.

From this one can conclude that only in water simulations, the first few eigenvectors contain significant information about protein correlated motions.



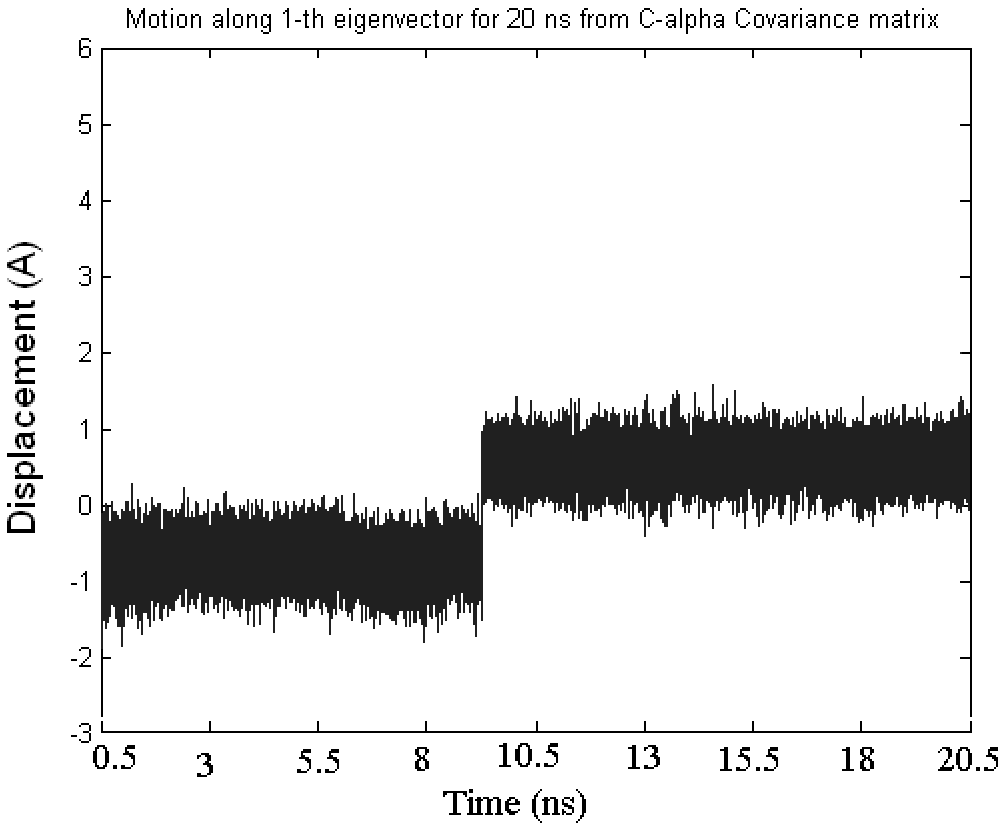
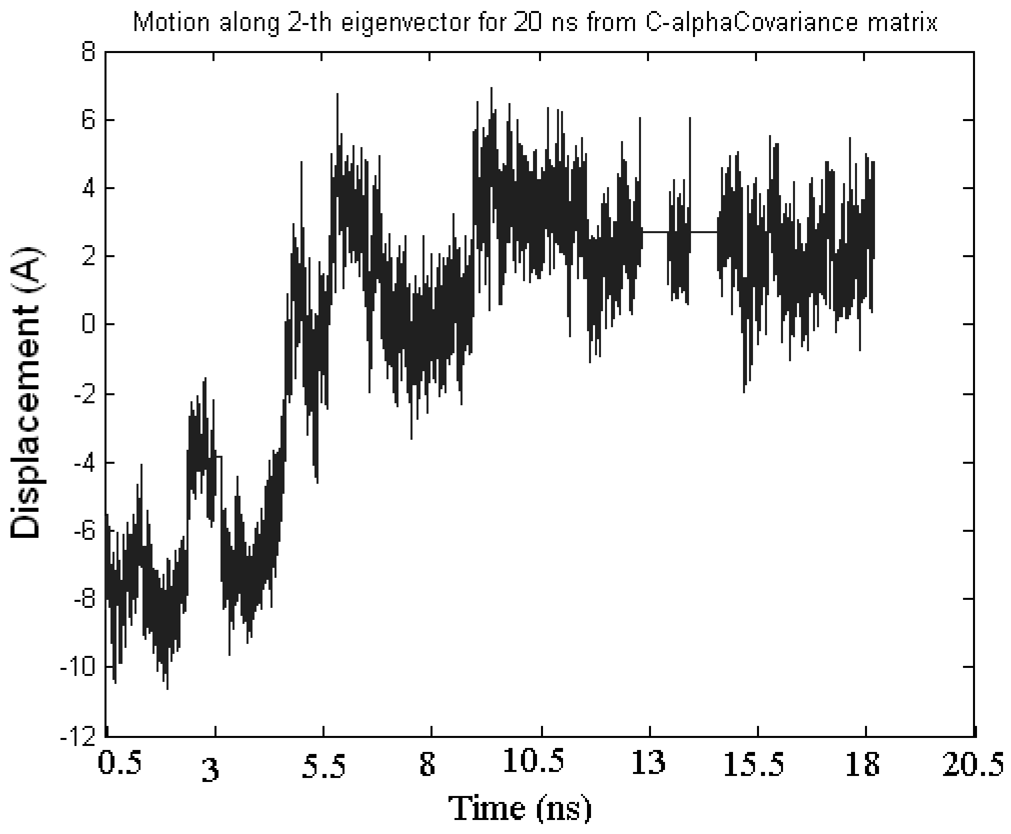


Figure 2.18 Motions of 20ns along first eigenvector obtained from the $C\alpha$ coordinates covariance matrix of the water simulation (Top) and the vacuum simulation (Bottom)



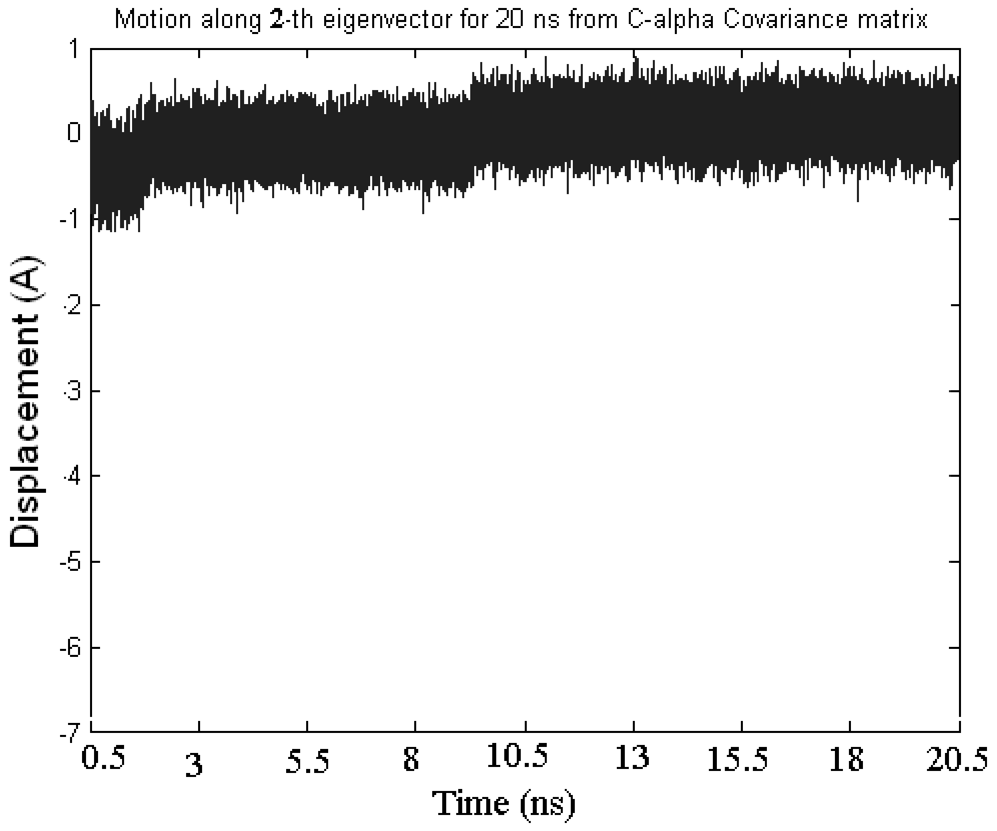
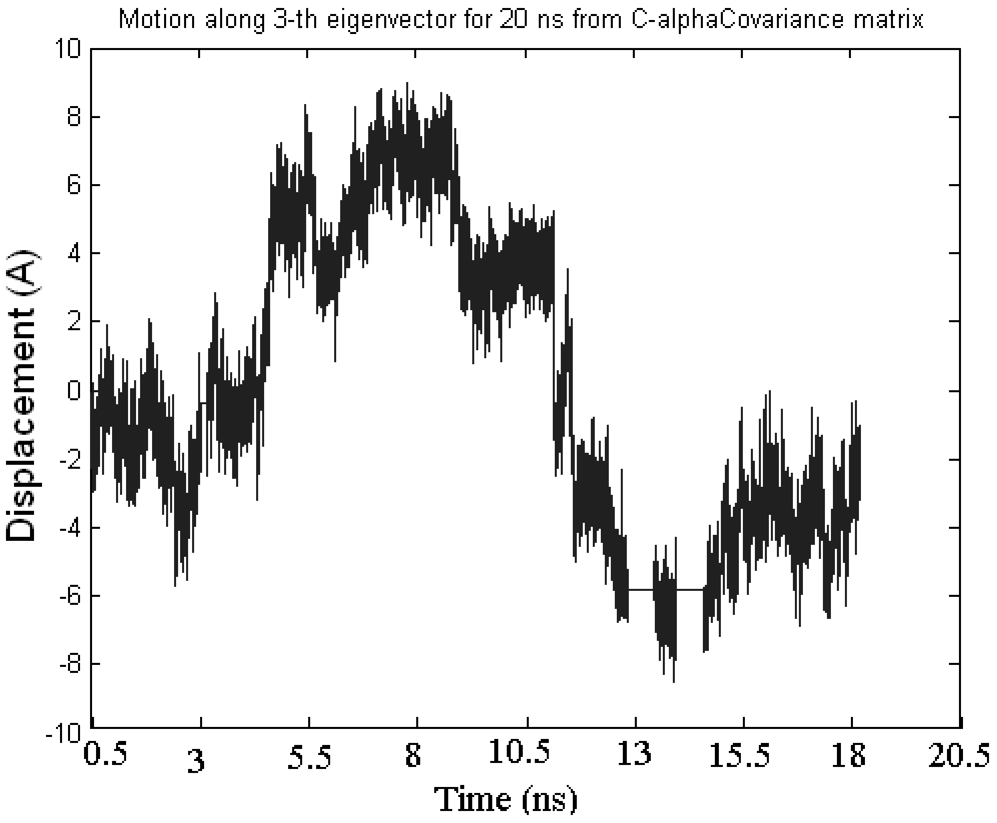


Figure 2.19 Motions of 20ns along second eigenvector obtained from the $C\alpha$ coordinates covariance matrix of the water simulation (Top) and the vacuum simulation (Bottom)



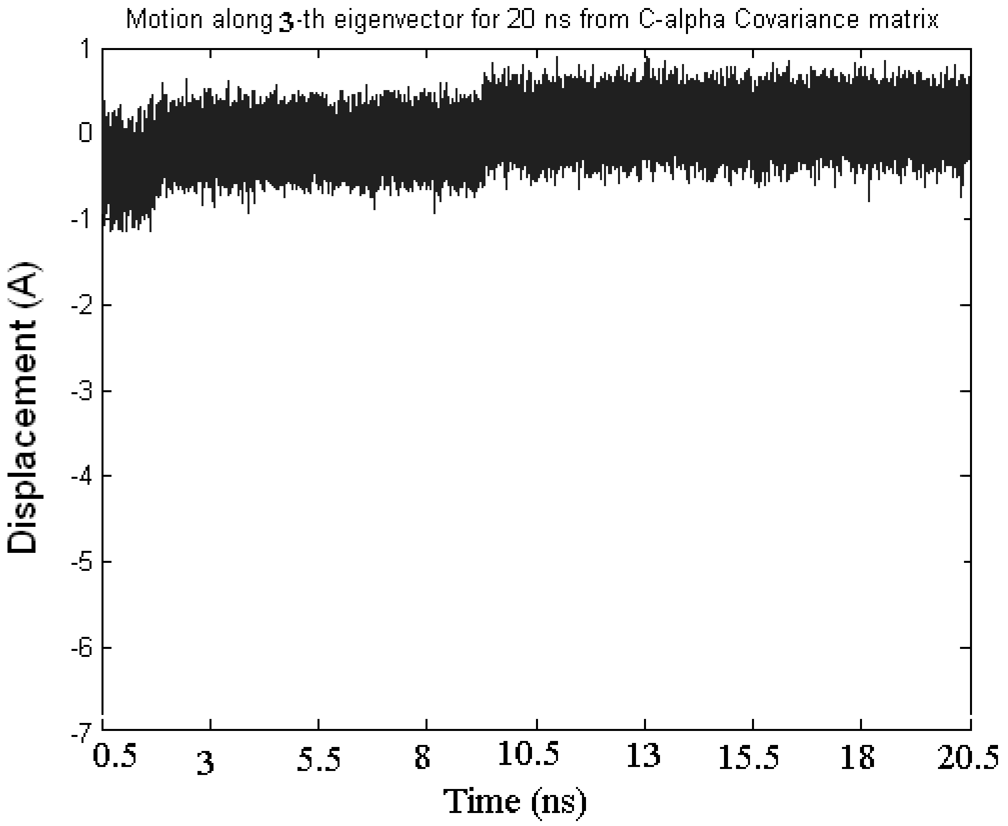
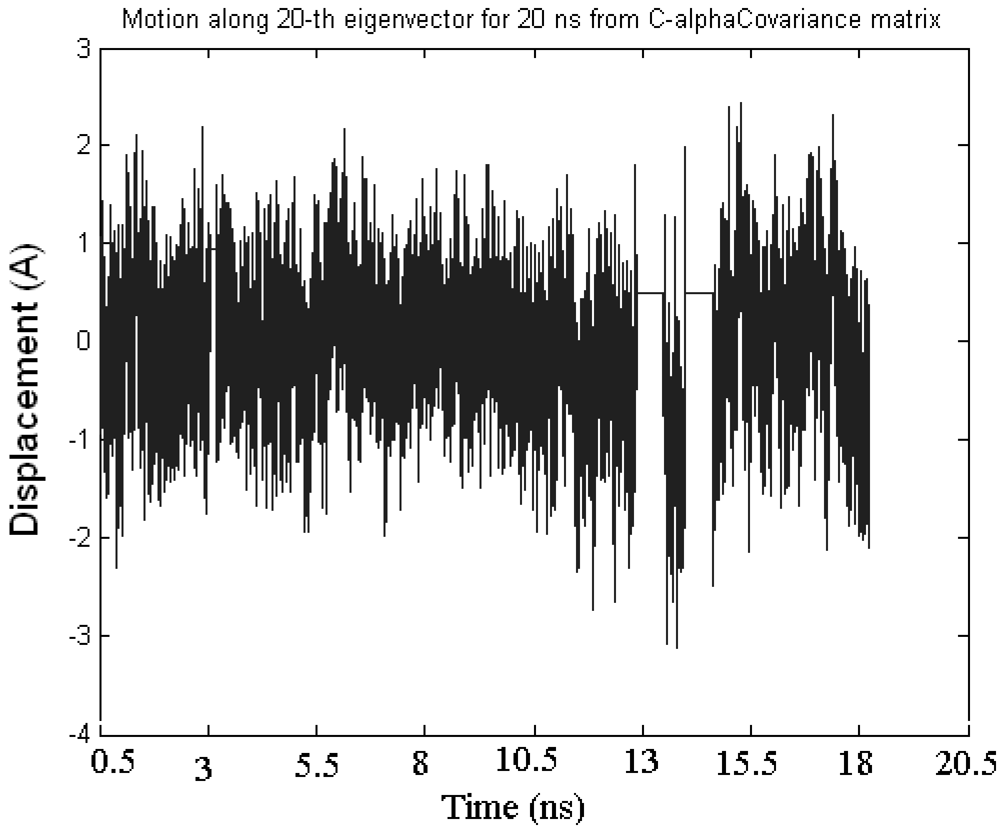


Figure 2.20 Motions of 20ns along third eigenvector obtained from the $C\alpha$ coordinates covariance matrix of the water simulation (Top) and the vacuum simulation (Bottom)



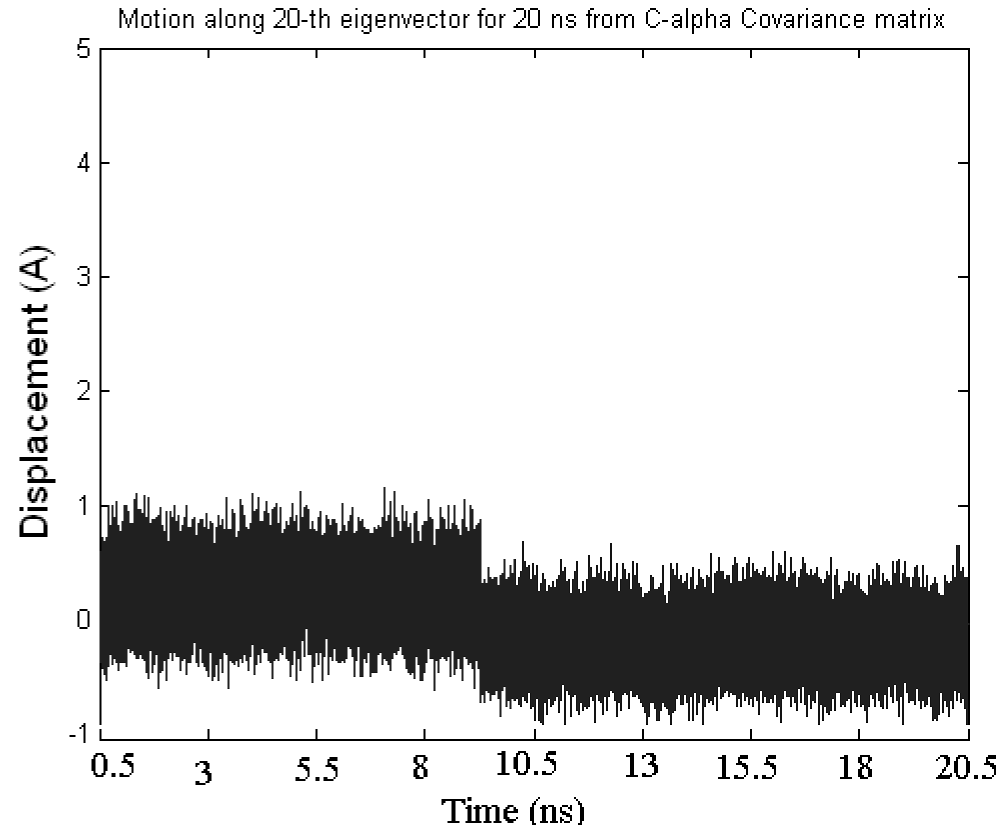
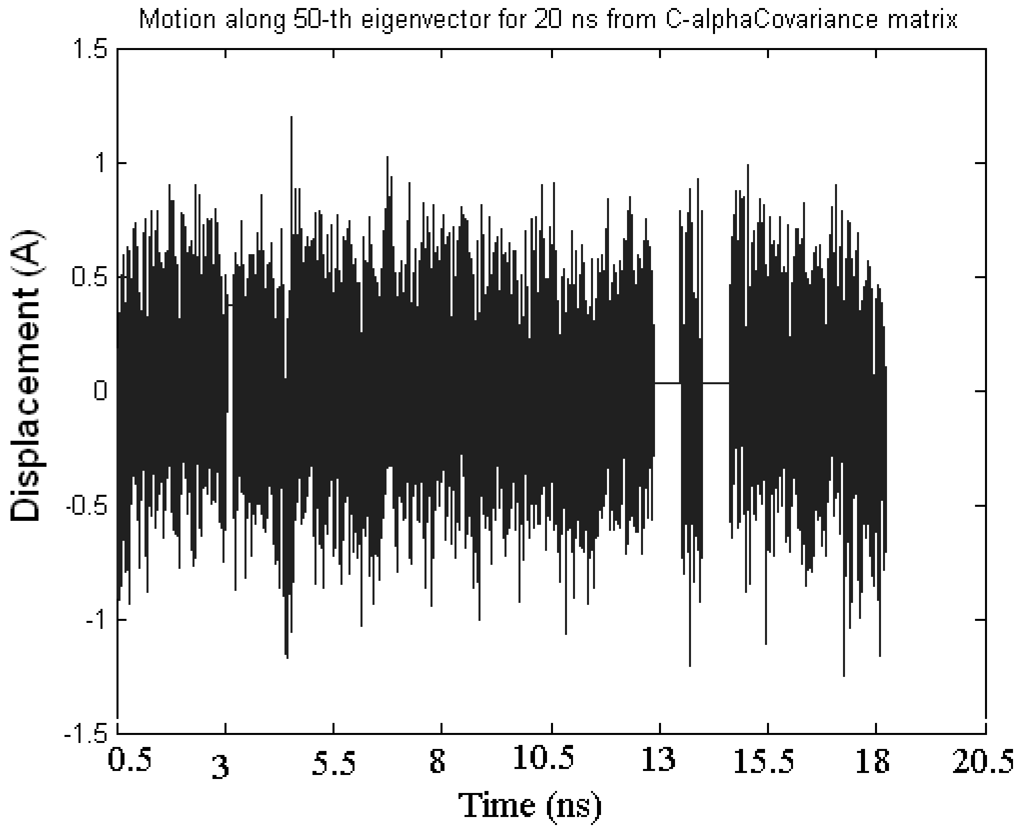


Figure 2.21 Motions of 20ns along twentieth eigenvector obtained from the $C\alpha$ coordinates covariance matrix of the water simulation (Top) and the vacuum simulation (Bottom)



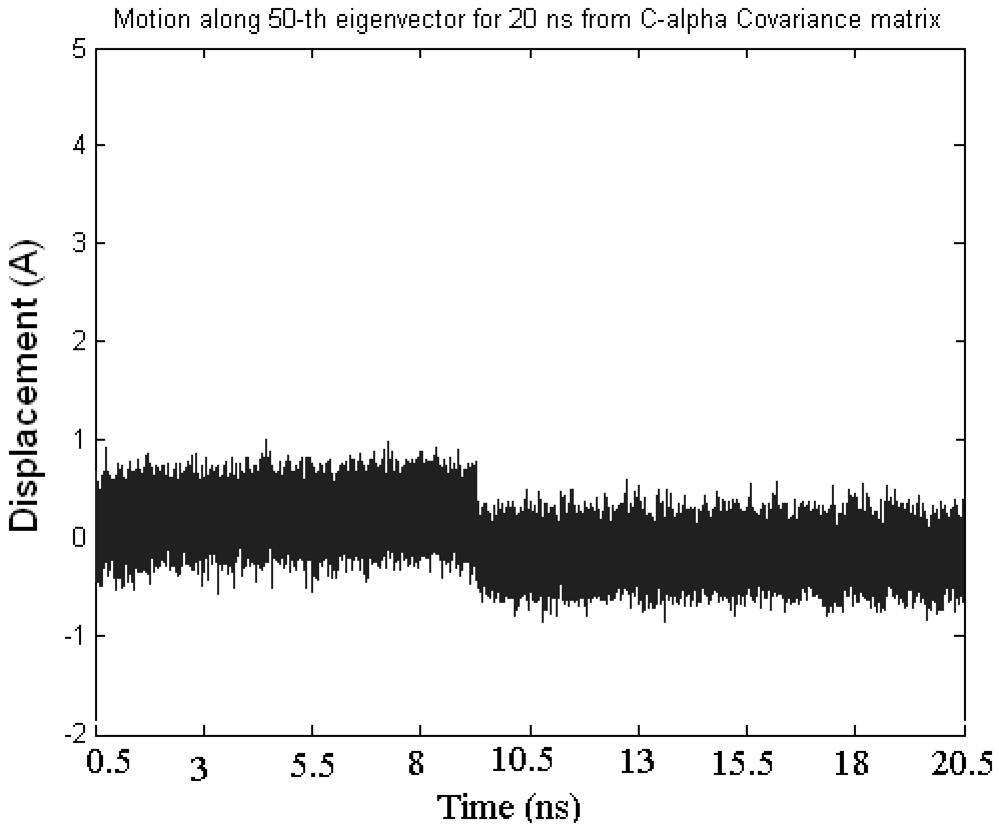


Figure 2.22 Motions of 20ns along fiftieth eigenvector obtained from the $C\alpha$ coordinates covariance matrix of the water simulation (Top) and the vacuum simulation (Bottom)

2.2.7 Motions Along Eigenvectors for each ns separately

It is preferable to be able to reduce the running time of a simulation to a minimum, and still get the same results one would get from a longer simulation. With this in mind, the 20 ns trajectory is divided into twenty 1 ns trajectories. For each of the 1 ns trajectories, a set of eigenvectors is calculated. The trajectory is then projected along the 1^{st}, 2^{nd}, 3^{rd}, and 50^{th} eigenvectors. This process is repeated twenty times, and the results are compared for each 1 ns. The question to be answered here is: Can a small part of the trajectory represent the whole trajectory when it comes to discussing the projection along eigenvectors? As can be seen by comparing Fig2.23A and Fig2.23B for example, it is clear that the projection of the first 1 ns trajectory along the eigenvectors is different than the projection of the 5^{th} 1 ns trajectory along the eigenvectors. However, if Fig2.23B, C, D are compared for the first eigenvector, a qualitatively similar behavior is seen. The projection value is steady and then increases sharply to a new steady state. Thus if the correlated motion along a certain eigenvector happens to have a time constant of 5 ns, a repeatable behavior will be seen. But interestingly enough, this does not apply to the projections at 20 ns.

In general, breaking up the trajectory into 1 ns pieces does not give the same results obtained from a 20 ns trajectory for all the motions. It is interesting to note further that the projections for the vacuum simulations along the eigenvectors consistently show random motion.

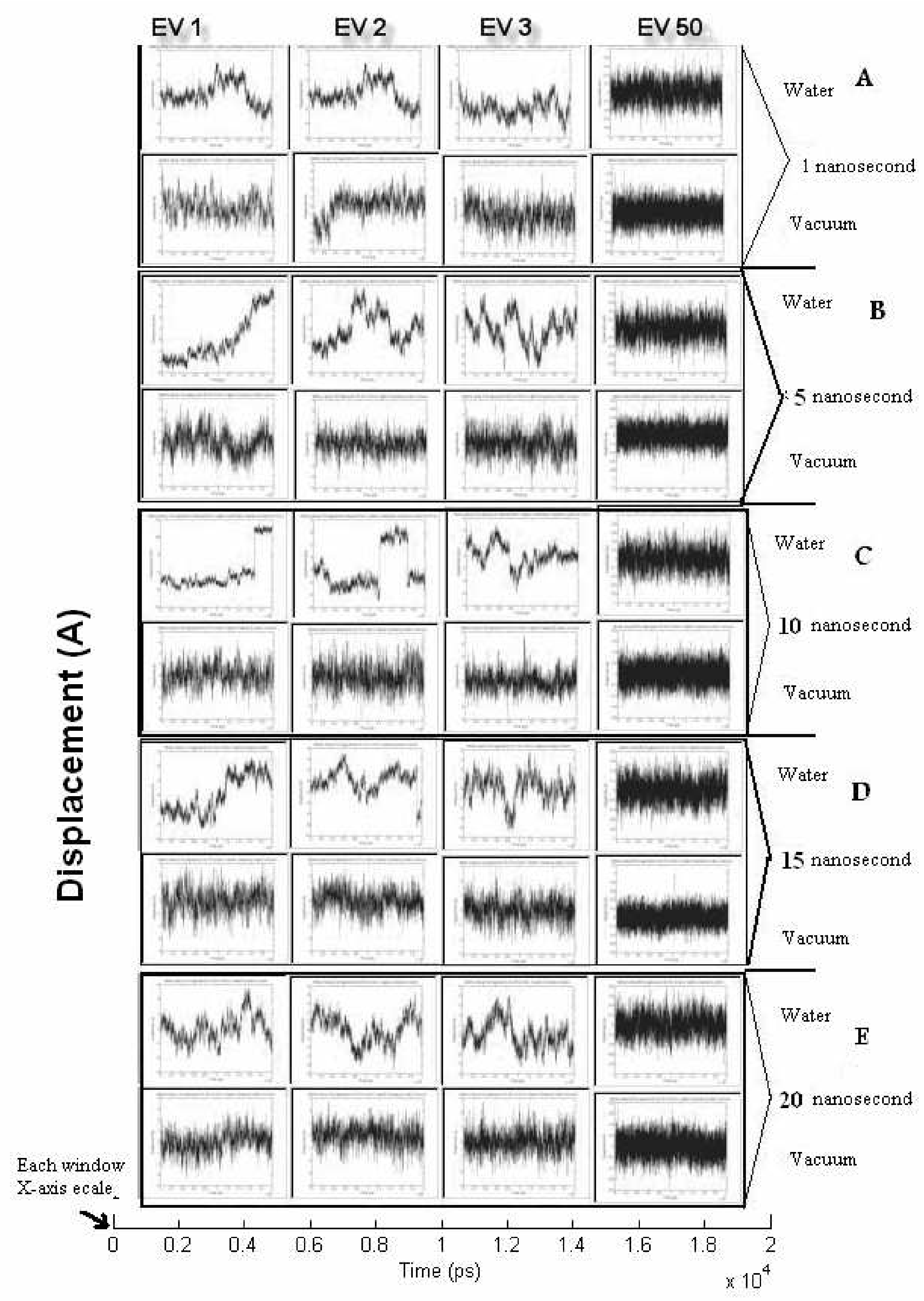
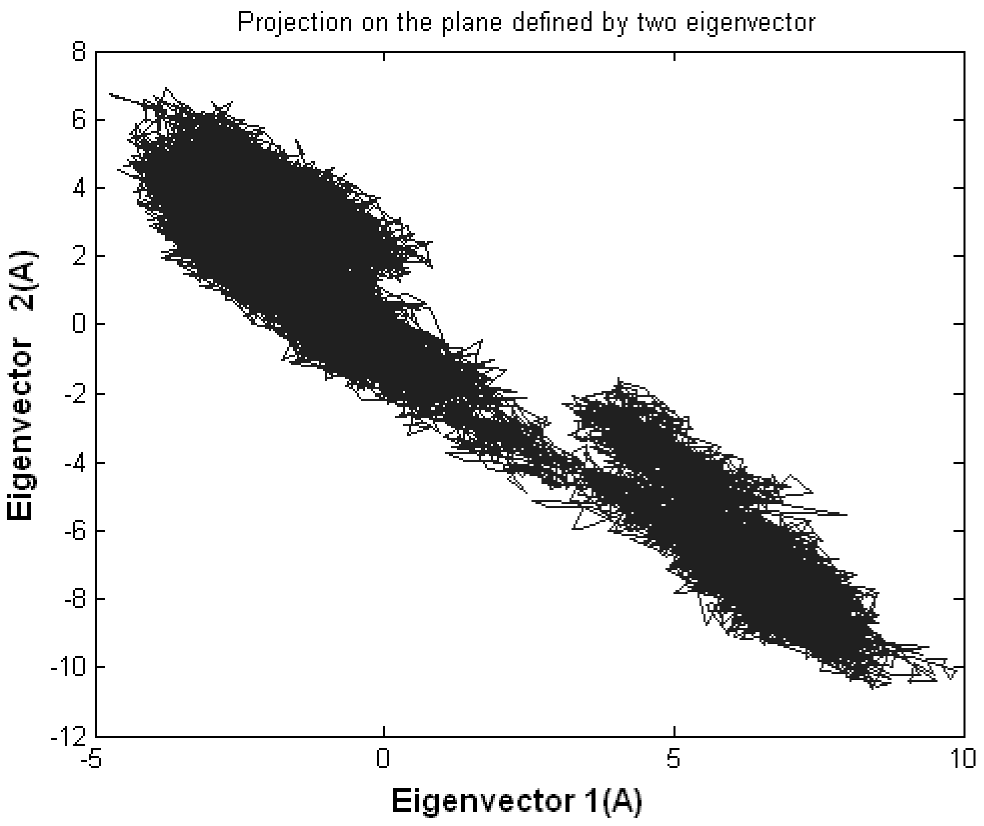


Figure 2.23 Motions along several eigenvectors (1,2,3,50) obtained from the $C\alpha$ 1ns coordinates covariance matrix of the water simulation (upper row) and the vacuum simulation (lower row), (A) 1ns motions after the first ns, (B) 1ns motions after the fifth ns (C) 1ns motions after the tenth ns (D) 1ns motions after the fifteenth ns (E) 1ns motions after the twentieth ns

2.2.8 Trajectory projected on planes

It is possible to project a trajectory along a plane defined by two eigenvectors. The trajectory is represented as a pair of coordinates on this plane. The two points represent the projection of the trajectory along the two different eigenvectors. By following the projection pair versus time, one can get a visual representation of where the protein spends significant amounts of time. This might be used to define a "state" of correlated motion. The Top of Figure (Fig2.25) shows the projection of the 20 ns trajectory on a plane defined by the first and second eigenvectors for the water simulation. One can discern two different states represented by the almost elliptical structures connected by a narrow strip. The bottom of Figure (Fig2.25) shows the projection of the 20 ns trajectory on a plane defined by the first and second eigenvectors for the vacuum simulation. Two different "states" are represented by the two circular patches. For comparison, the top and bottom columns of Figure (Fig2.30) show the projection of the 20 ns simulation along a plane defined by the twentieth and fiftieth eigenvectors for the water and the vacuum simulations respectively. One can discern two circular patches that basically hover around zero. Both cases represent random motion that shows no correlation. A similar behavior occurs when the trajectory is projected along non-significant eigenvectors.



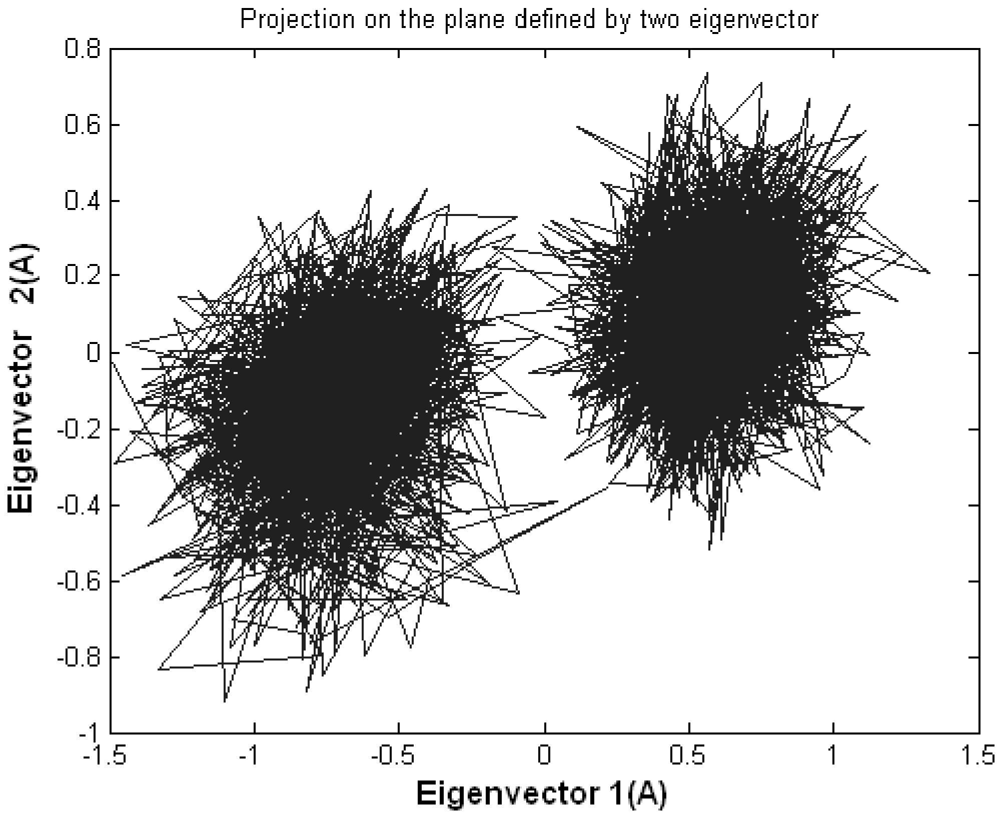
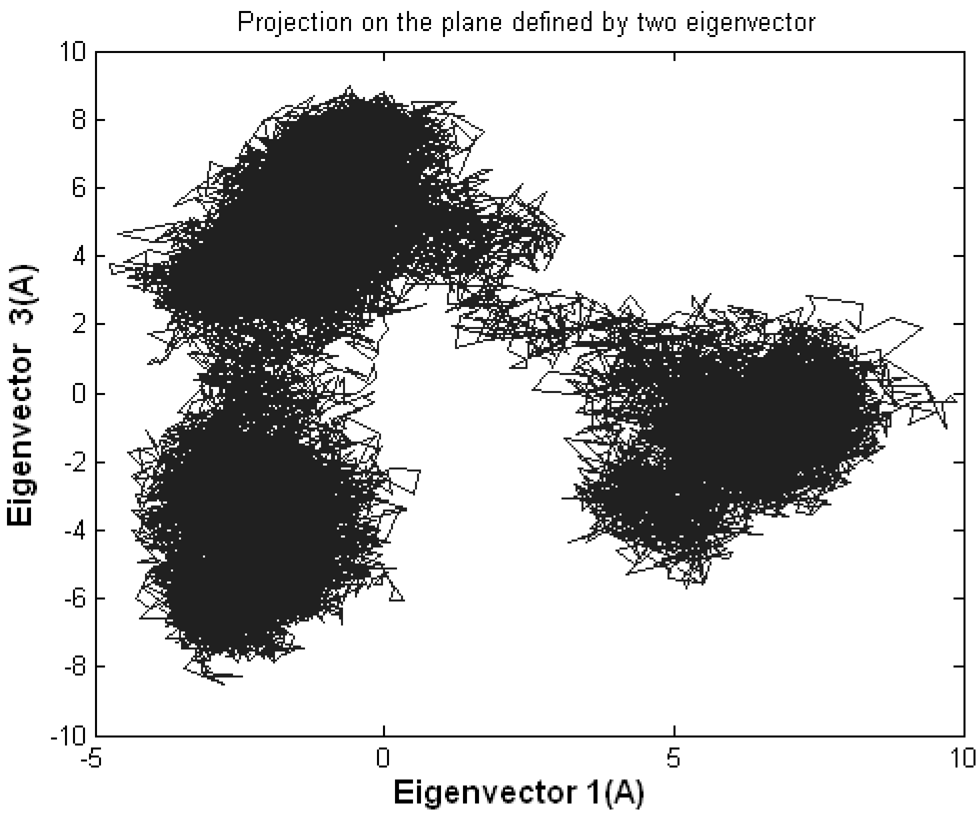


Figure 2.24 Projection of the trajectory (water simulation in Top and vacuum simulation Bottom) on the planes defined by two eigenvectors from the $C\alpha$ coordinates covariance matrix. (A) Horizontal axis: displacement along first eigenvector. Vertical axis: displacement along second eigenvector.



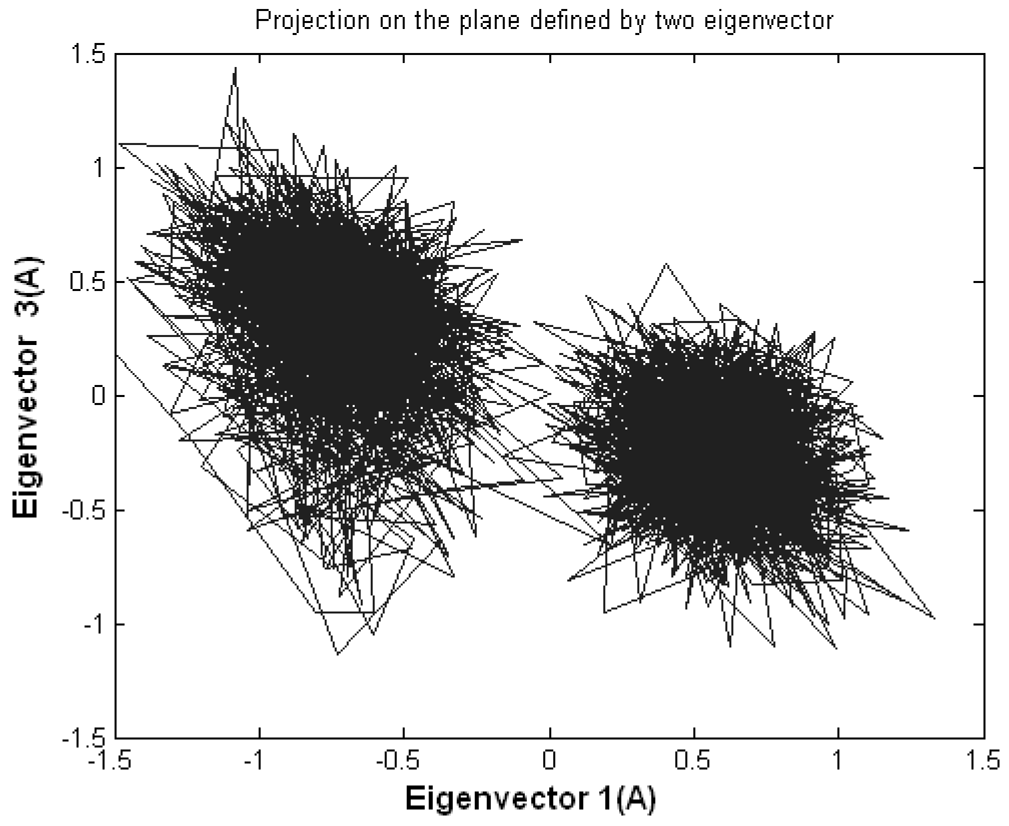
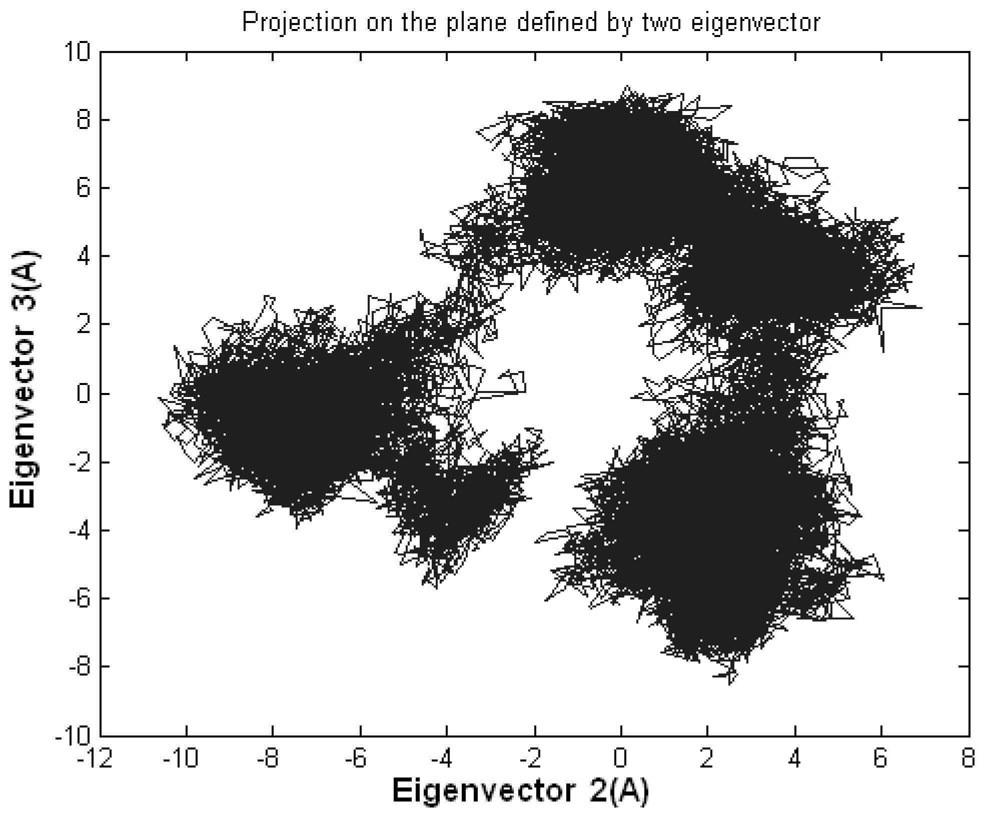


Figure 2.25 Projection of the trajectory (water simulation in Top and vacuum simulation Bottom) on the planes defined by two eigenvectors from the $C\alpha$ coordinates covariance matrix. (A) Horizontal axis: displacement along first eigenvector. Vertical axis: displacement along third eigenvector.



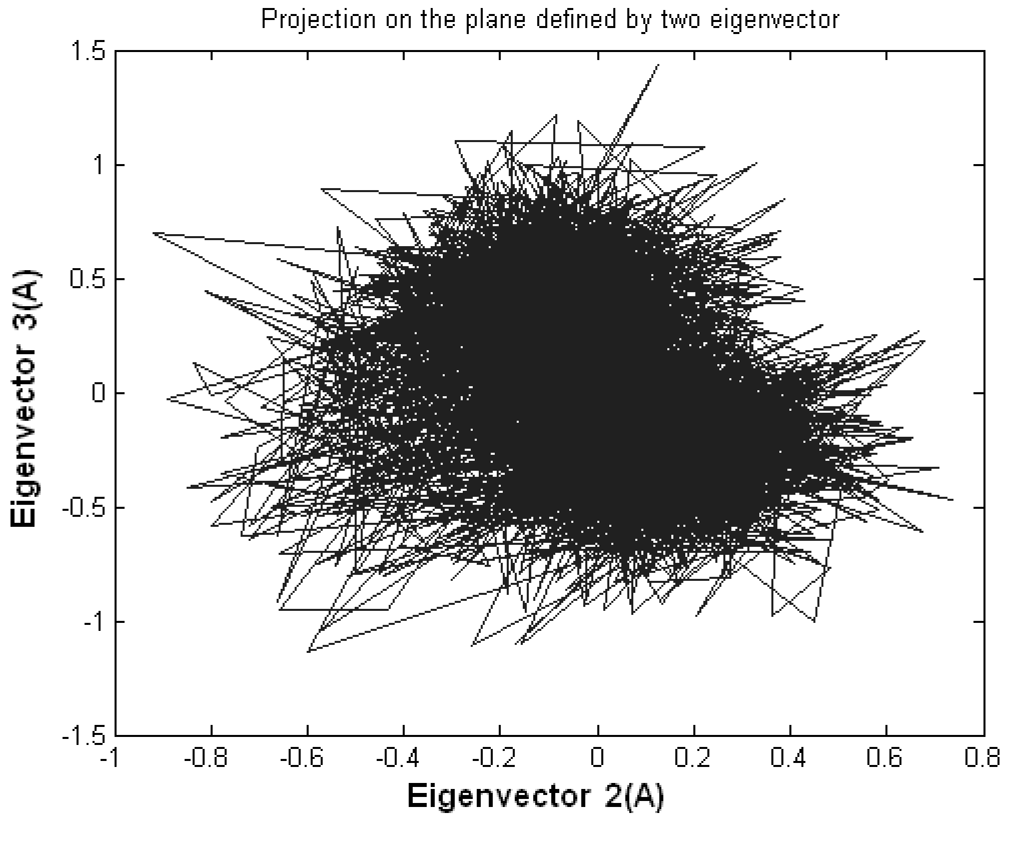
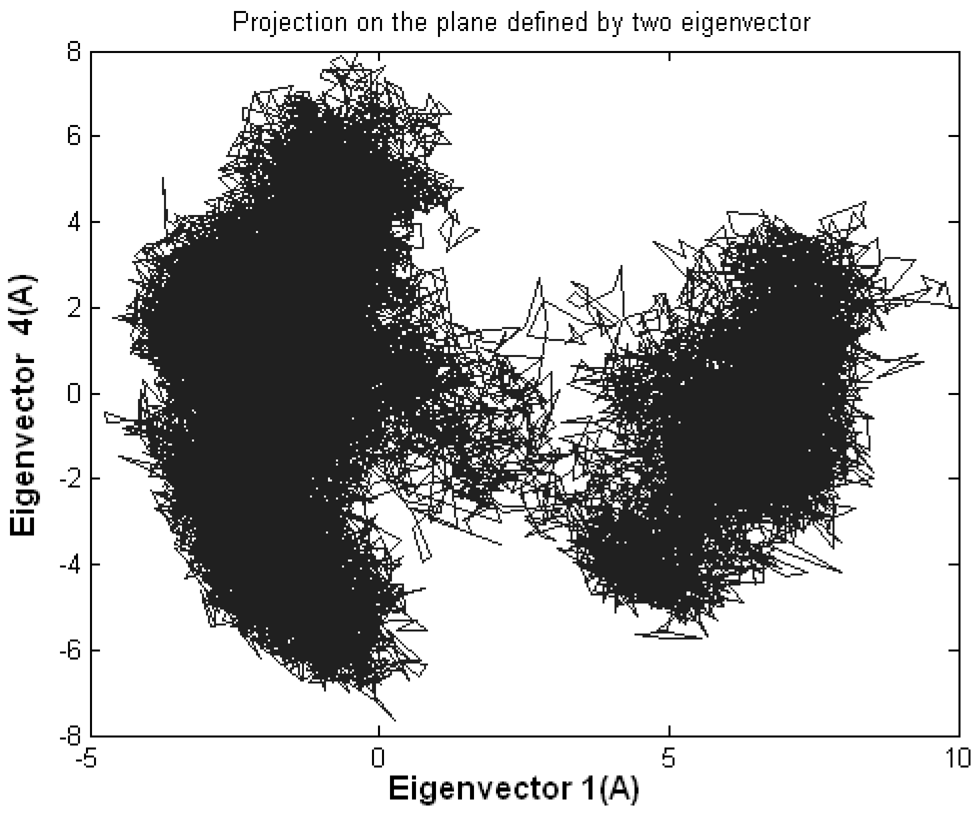


Figure 2.26 Projection of the trajectory (water simulation in Top and vacuum simulation Bottom) on the planes defined by two eigenvectors from the $C\alpha$ coordinates covariance matrix.(A) Horizontal axis: displacement along second eigenvector. Vertical axis: displacement along third eigenvector.



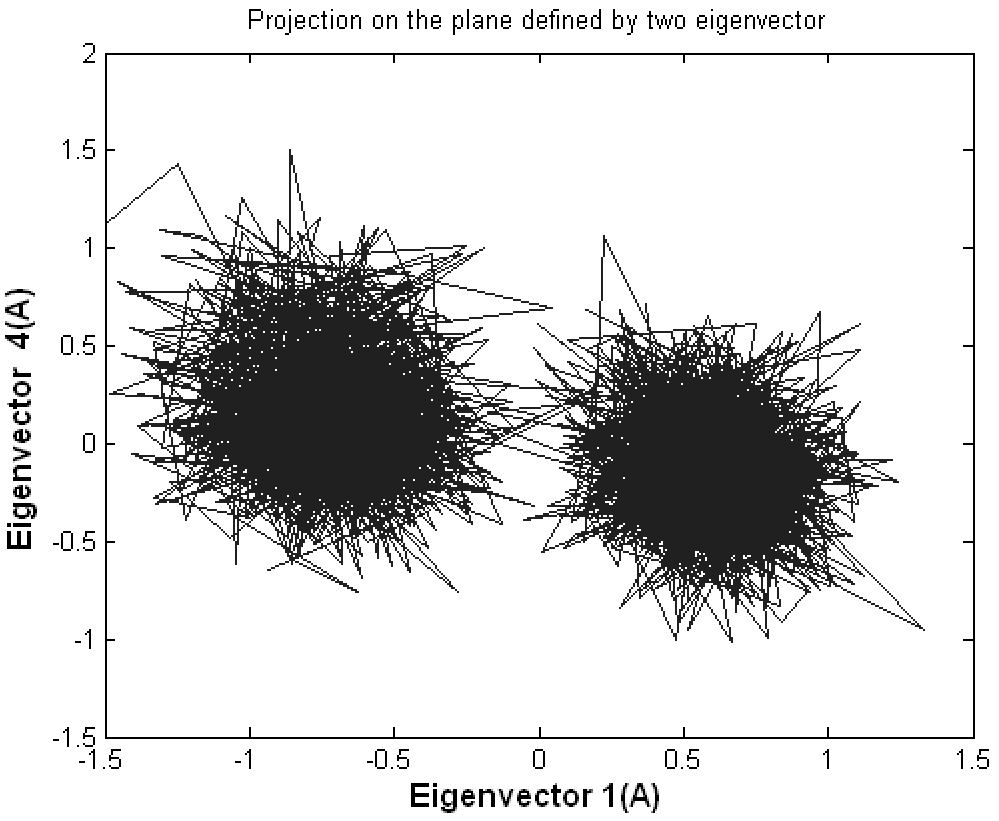
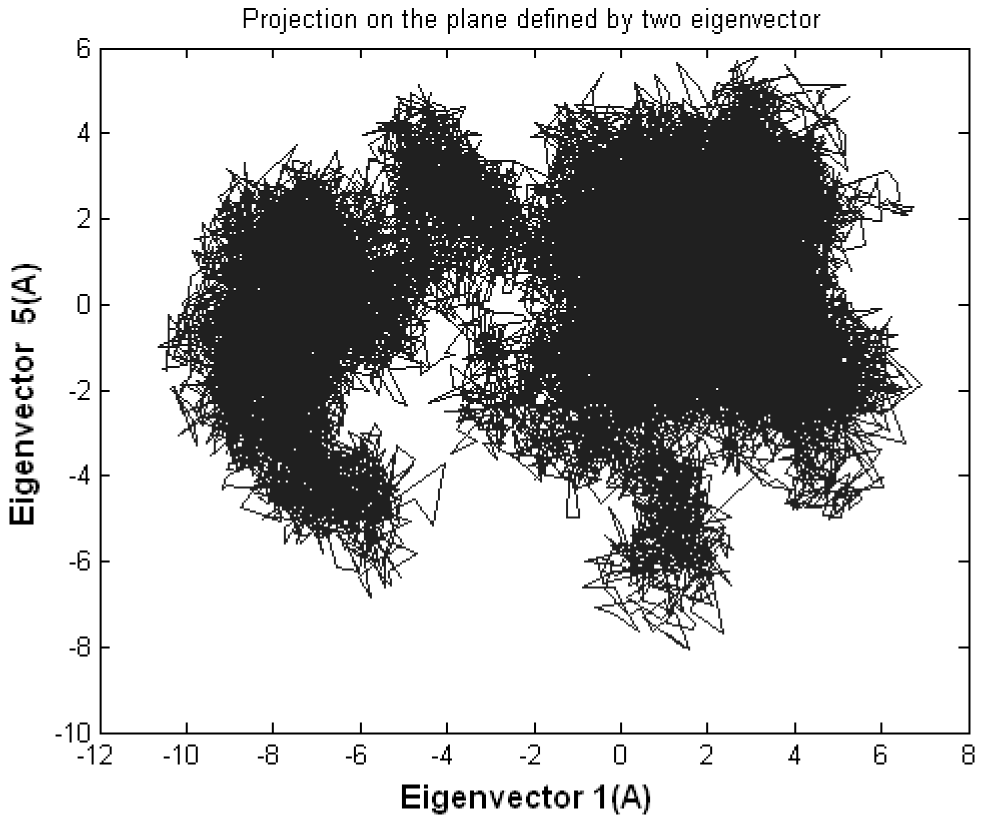


Figure 2.27 Projection of the trajectory (water simulation in Top and vacuum simulation Bottom) on the planes defined by two eigenvectors from the $C\alpha$ coordinates covariance matrix.(A) Horizontal axis: displacement along first eigenvector. Vertical axis: displacement along fourth eigenvector.



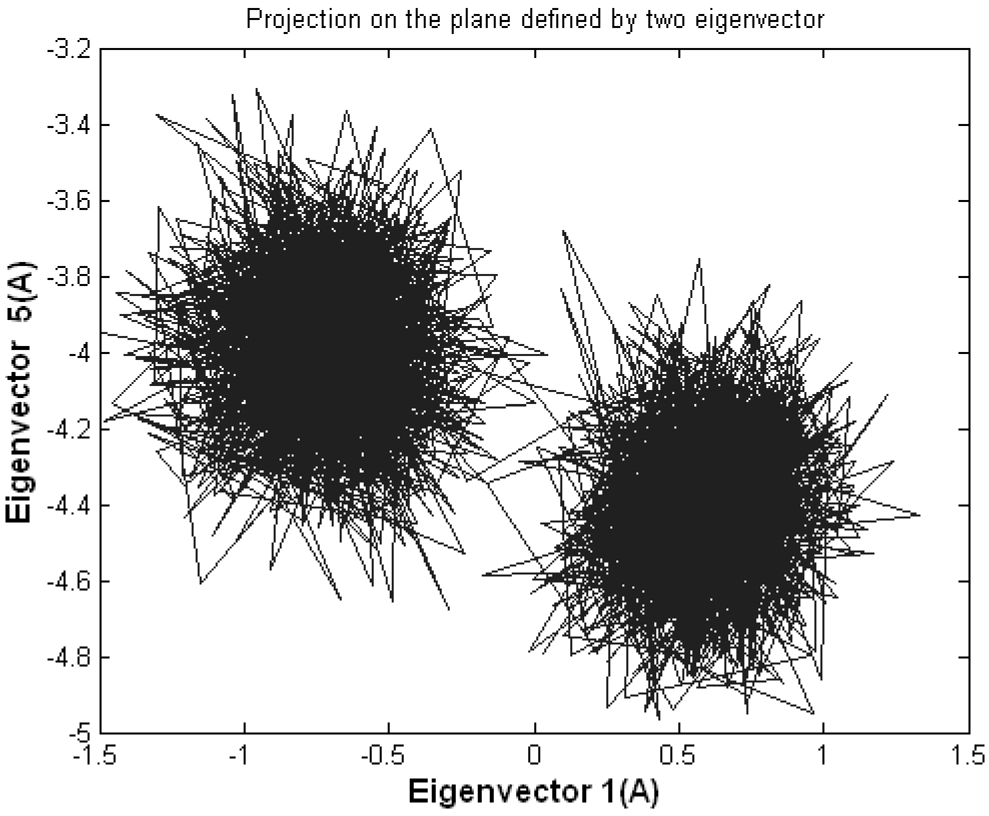
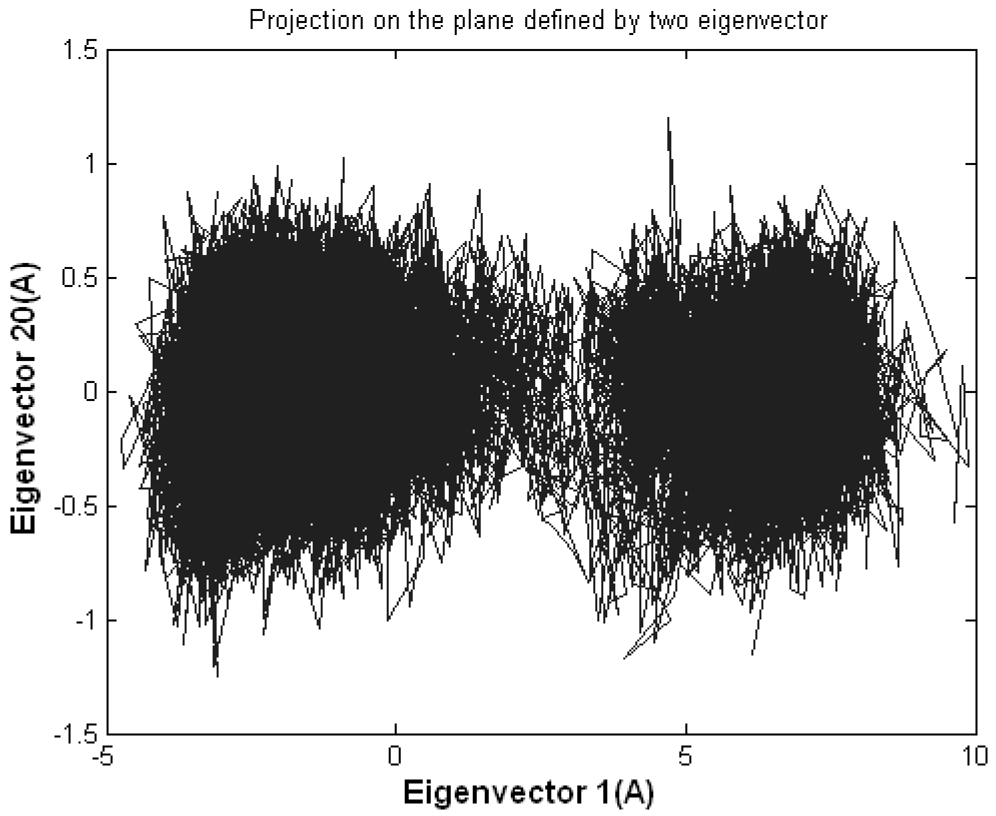


Figure 2.28 Projection of the trajectory (water simulation in Top and vacuum simulation Bottom) on the planes defined by two eigenvectors from the $C\alpha$ coordinates covariance matrix.(A) Horizontal axis: displacement along first eigenvector. Vertical axis: displacement along fifth eigenvector.



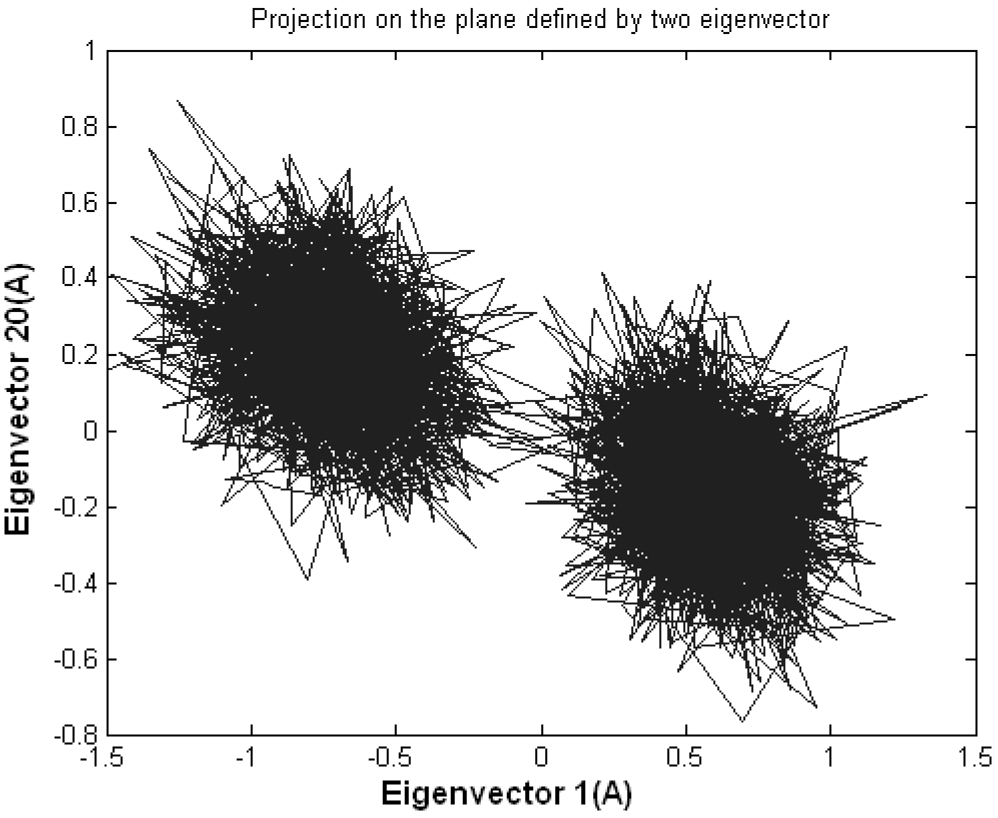
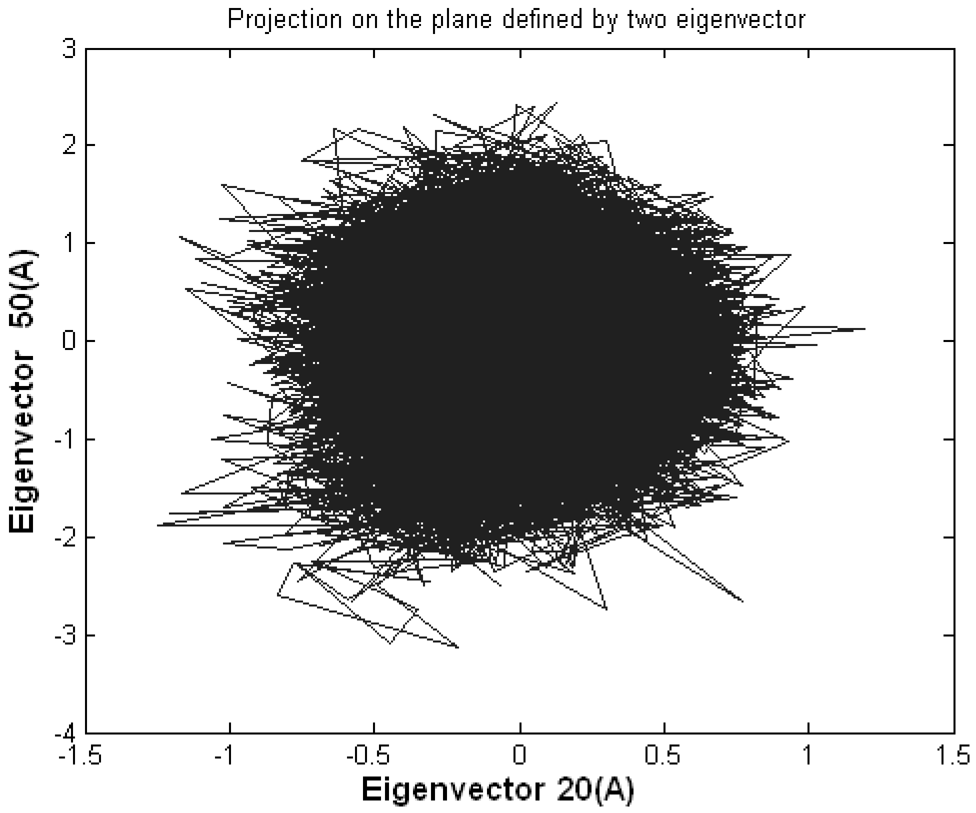


Figure 2.29 Projection of the trajectory (water simulation in Top and vacuum simulation Bottom) on the planes defined by two eigenvectors from the $C\alpha$ coordinates covariance matrix.(A) Horizontal axis: displacement along first eigenvector. Vertical axis: displacement along twentieth eigenvector.



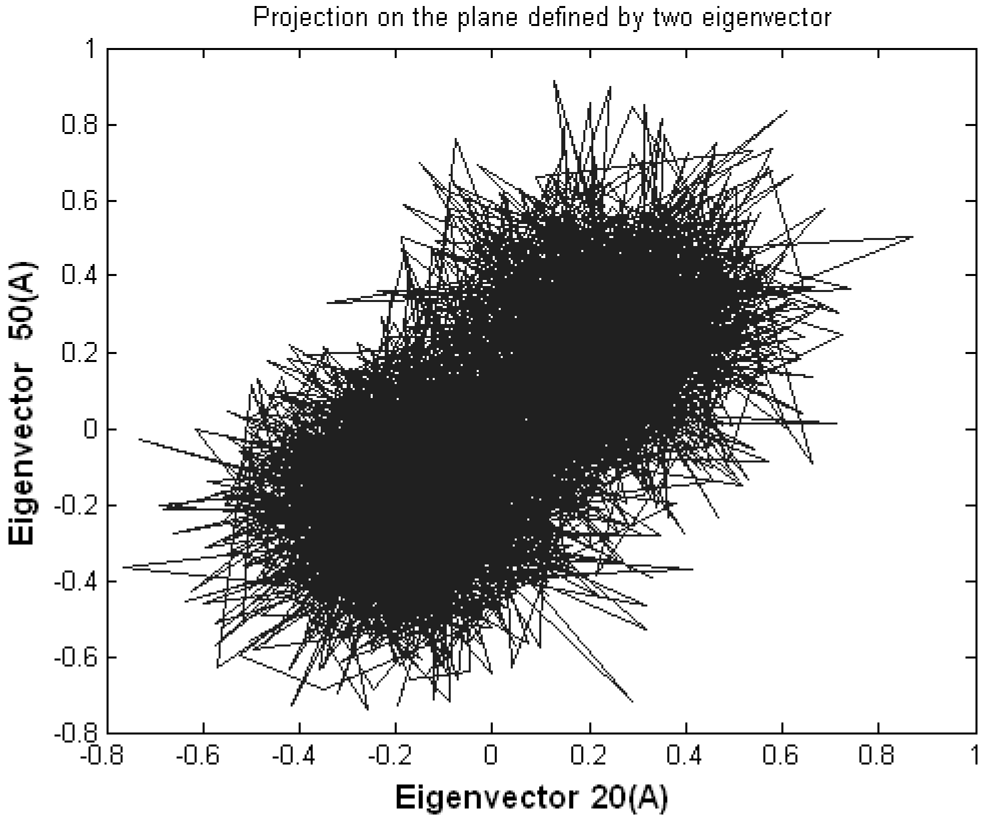
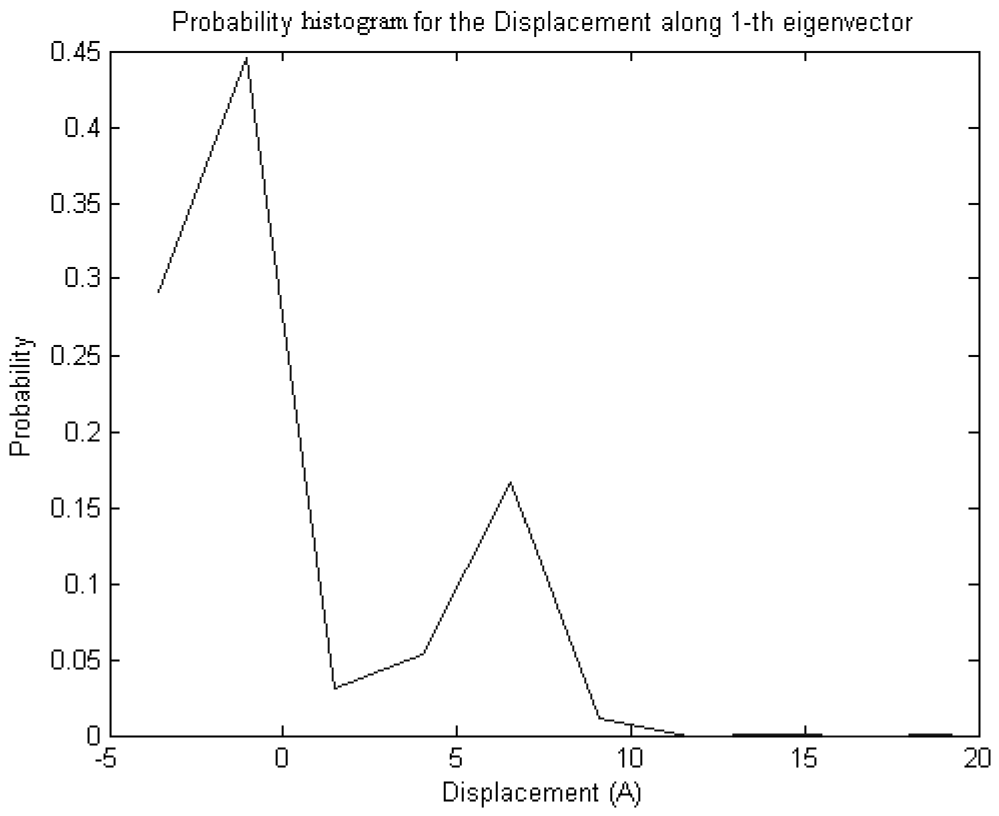


Figure 2.30 Projection of the trajectory (water simulation in Top and vacuum simulation Bottom) on the planes defined by two eigenvectors from the $C\alpha$ coordinates covariance matrix. (A) Horizontal axis: displacement along twentieth eigenvector. Vertical axis: displacement along fiftieth eigenvector.

2.2.9 Probability distributions of the Motions

Figures (Fig2.31, Fig2.32, Fig2.33, Fig2.34, Fig2.35) shows the distribution functions for the displacements along a number of significant eigenvectors in water (top of Fig2.31, Fig2.32, Fig2.33, Fig2.34, Fig2.35) and vacuum (bottom of Fig2.31, Fig2.32, Fig2.33, Fig2.34, Fig2.35). Notice that the non-Gaussian distributions belong to the first eigenvectors. This attests to the random nature of motion along the higher eigenvectors. In other words, all significant correlated motions are described by the first few eigenvectors. This behavior is more pronounced in the water simulation.



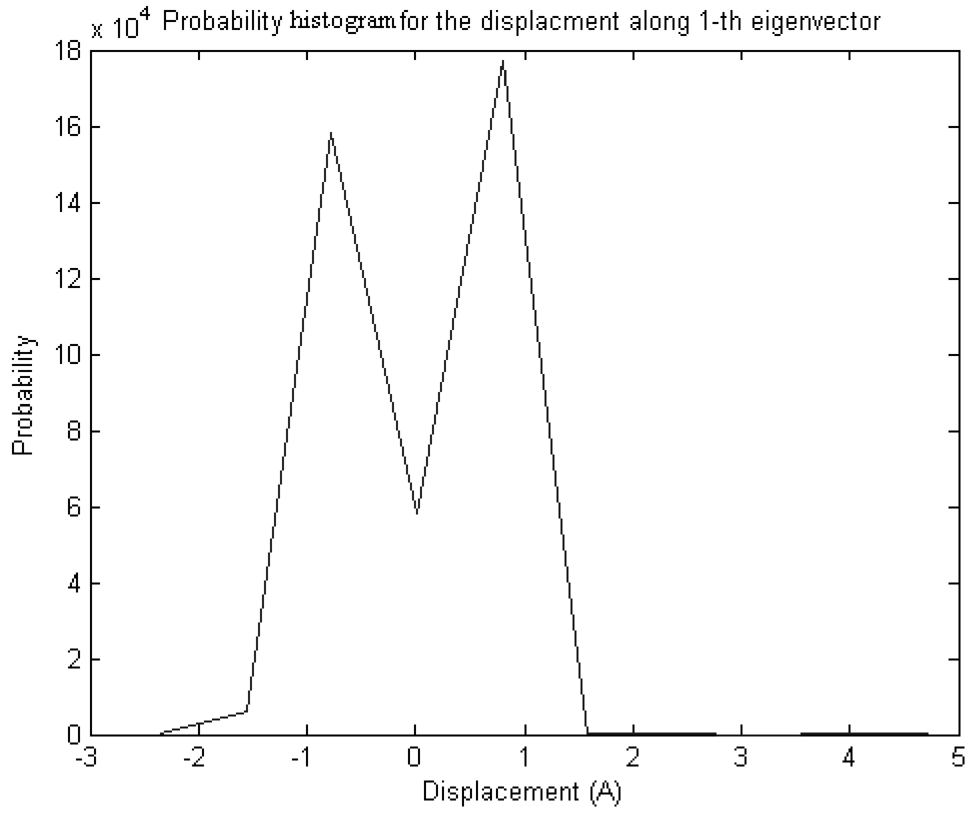
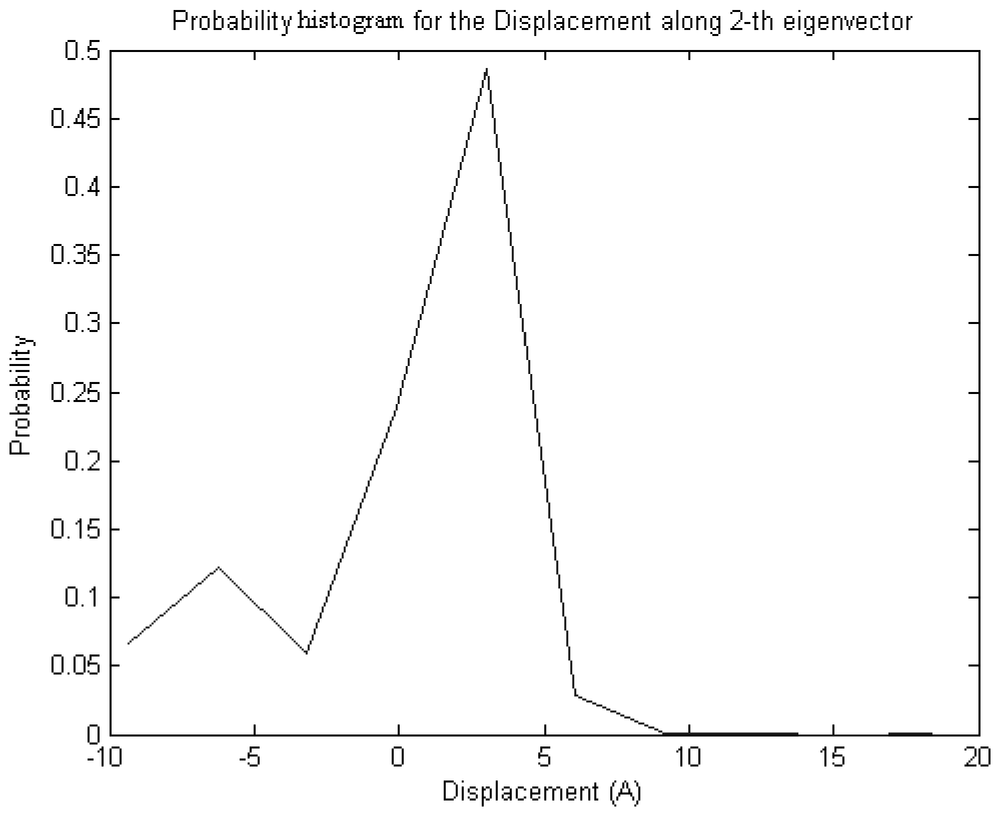


Figure 2.31 Probability distributions for the displacements along first eigenvectors for both the water simulation (Top) and the vacuum simulation (Bottom).



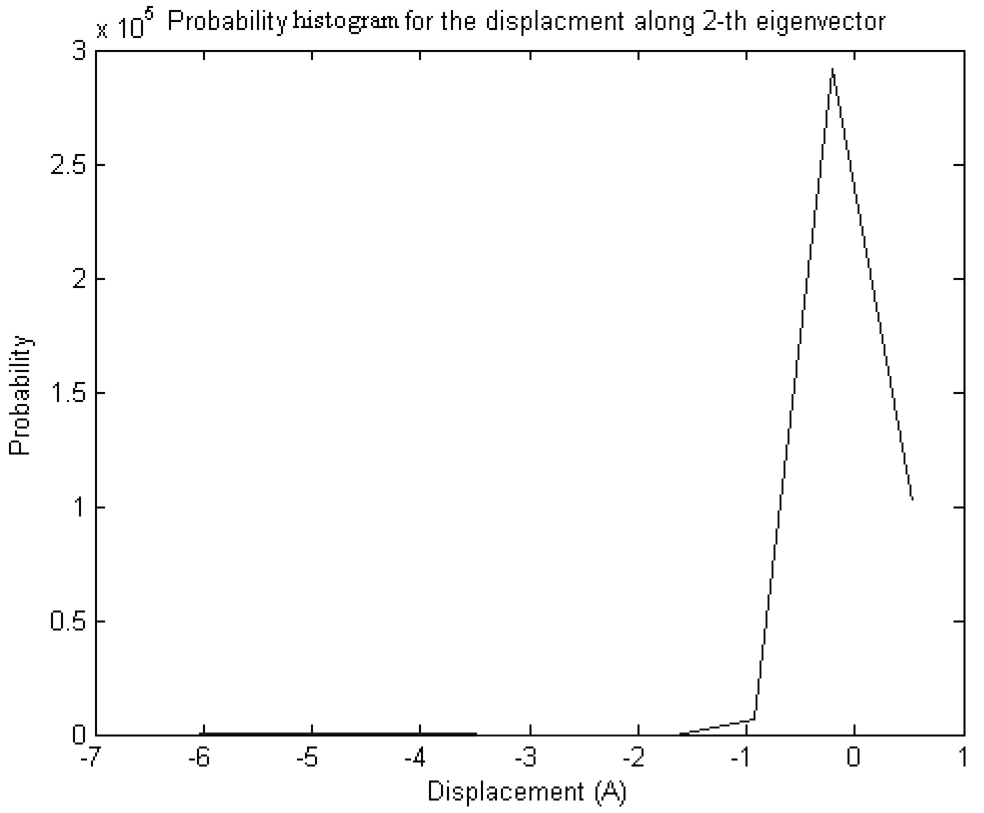
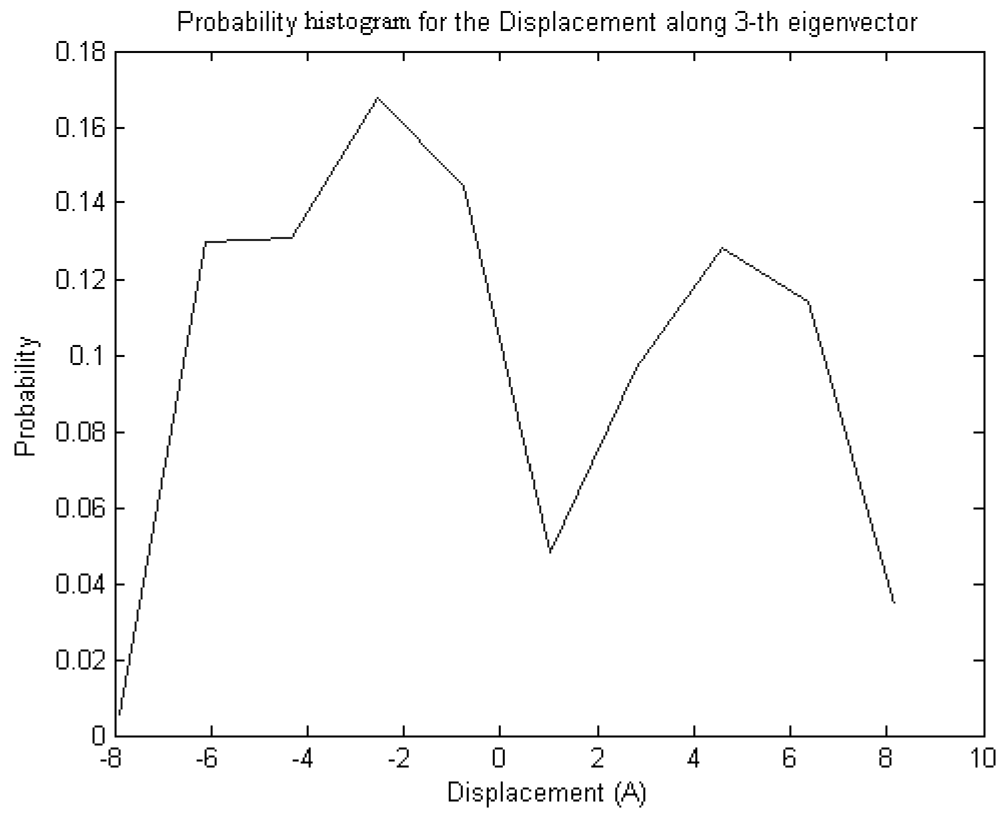


Figure 2.32 Probability distributions for the displacements along second eigenvectors for both the water simulation (Top) and the vacuum simulation (Bottom).



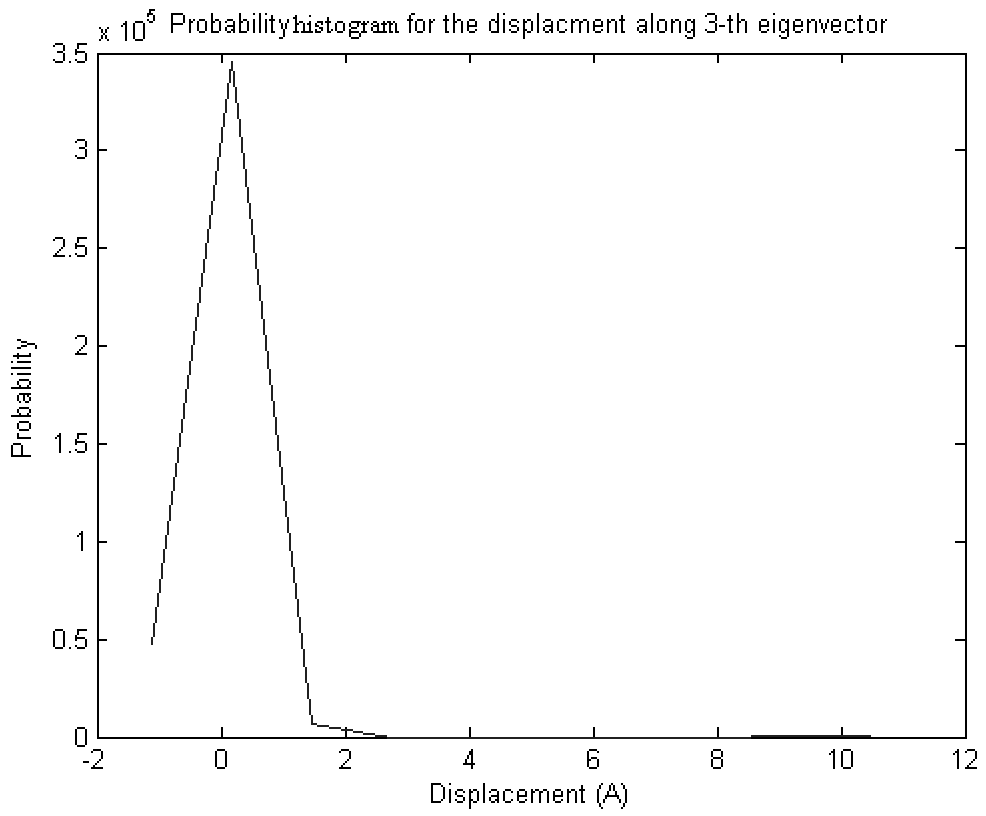
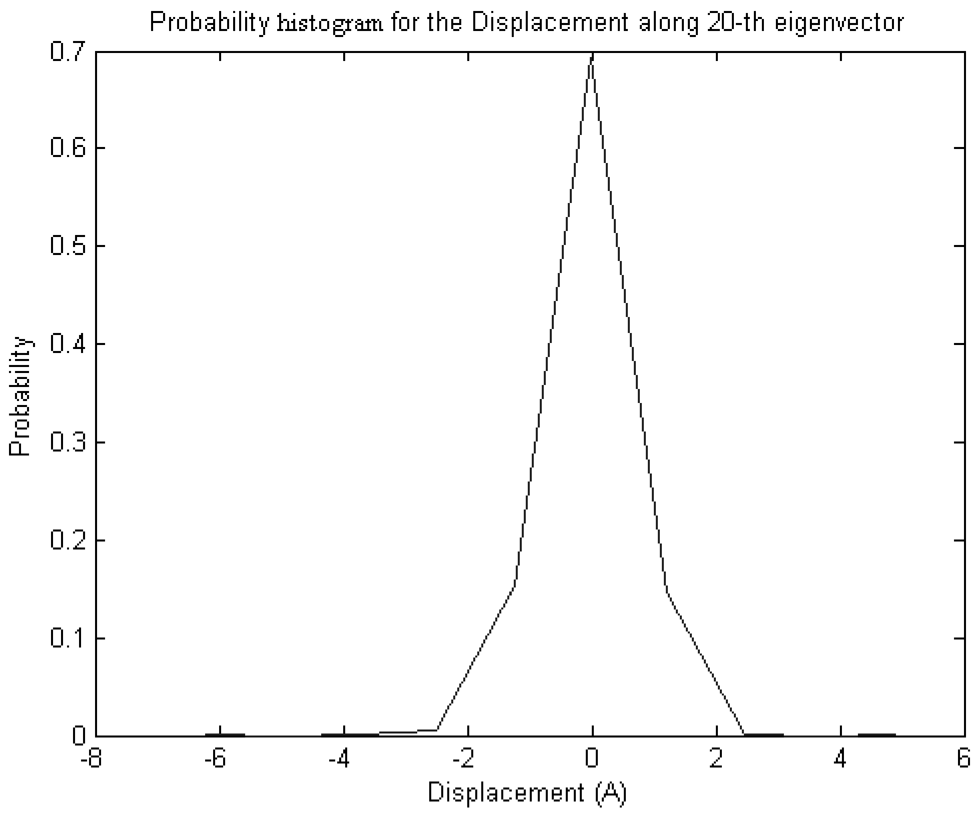


Figure 2.33 Probability distributions for the displacements along third eigenvectors for both the water simulation (Top) and the vacuum simulation (Bottom).



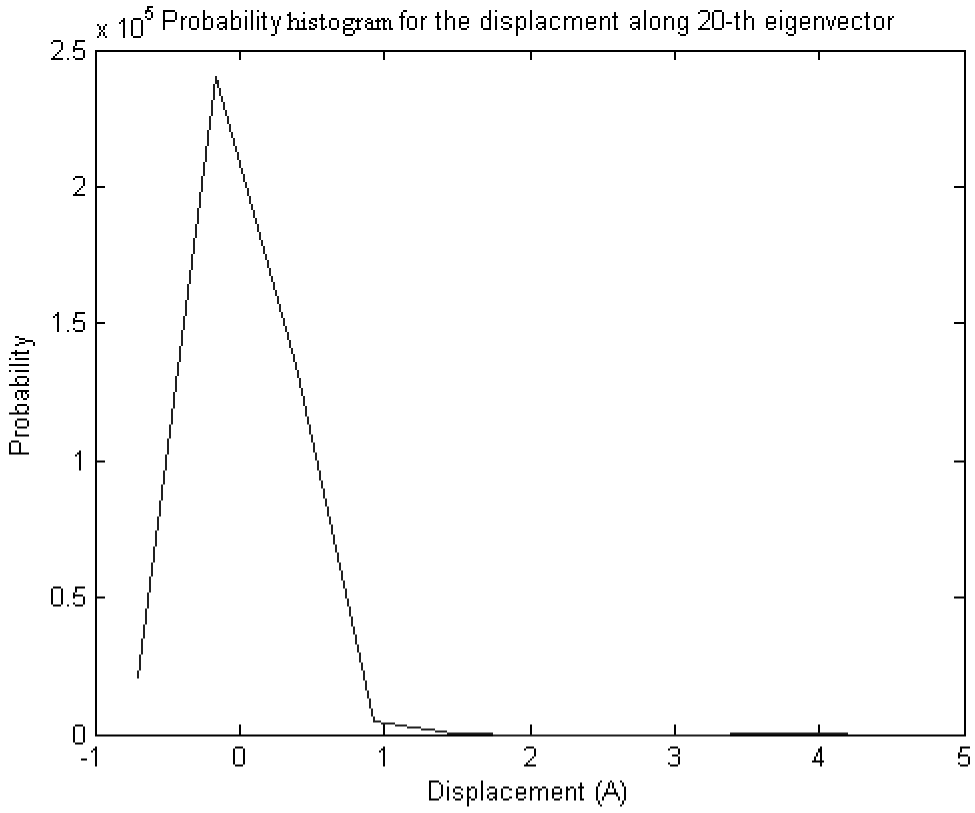
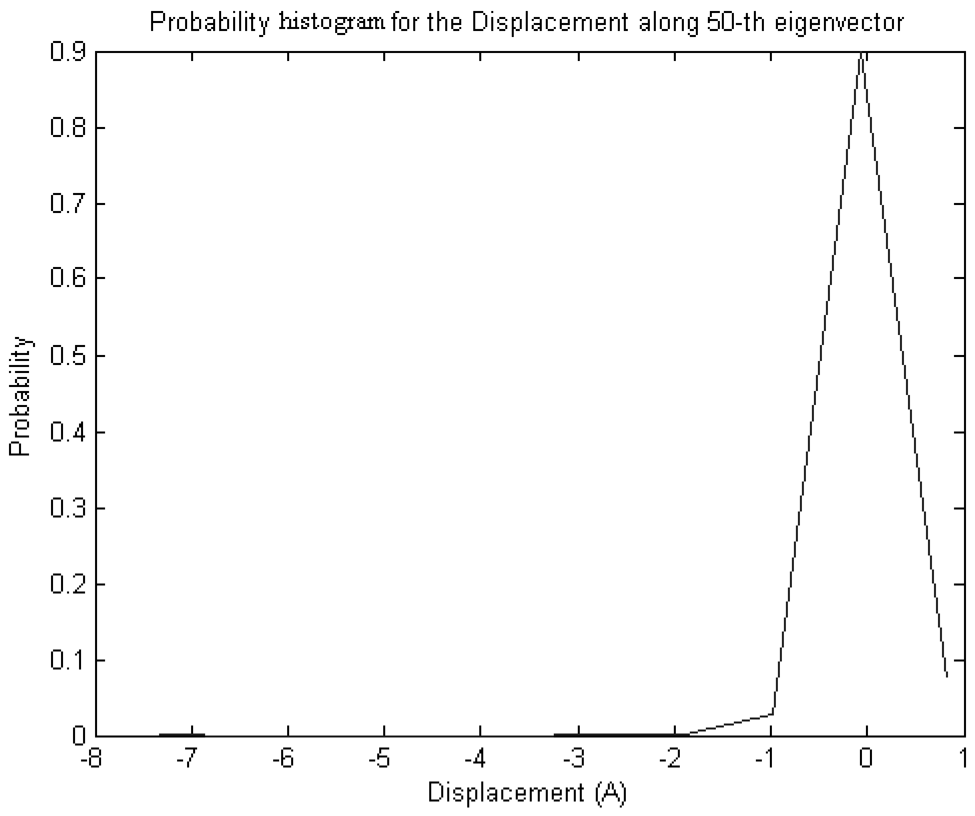


Figure 2.34 Probability distributions for the displacements along twentieth eigenvectors for both the water simulation (Top) and the vacuum simulation (Bottom).



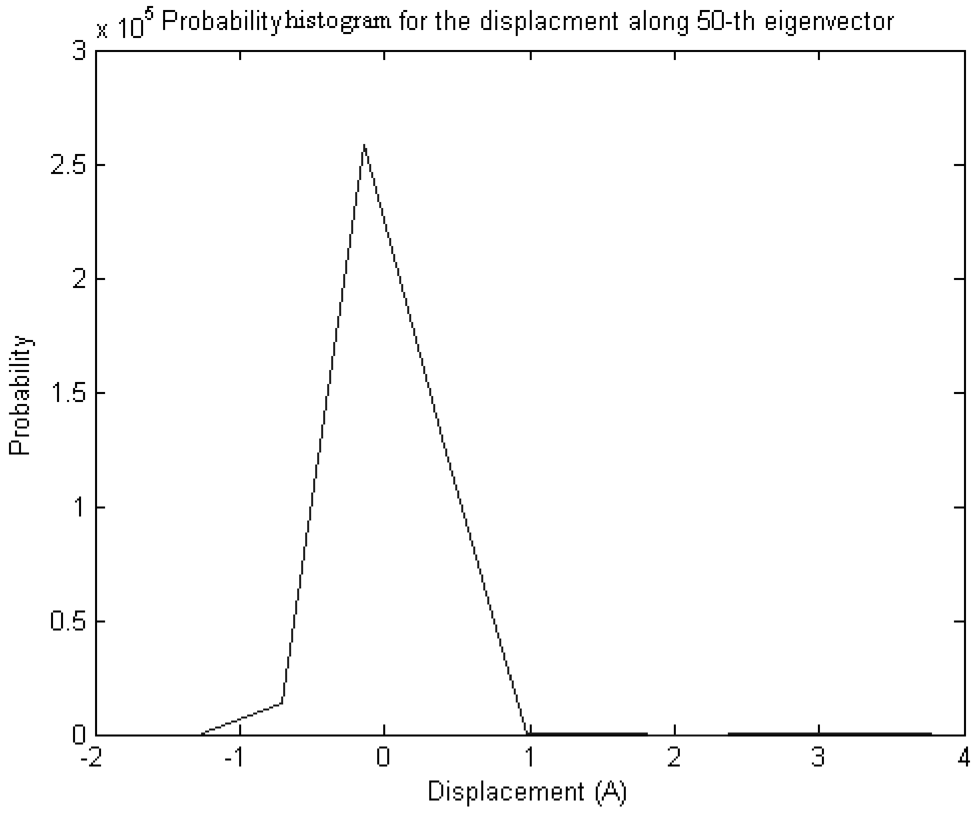


Figure 2.35 Probability distributions for the displacements along fiftieth eigenvectors for both the water simulation (Top) and the vacuum simulation (Bottom).

2.2.10 Projecting several separate eigenvectors onto first ns

The set of eigenvectors evaluated for each nanosecond separately was projected on the set of eigenvectors evaluated from the first nanosecond. The idea is to check for repeatability in the eigenvector sets. These projections are represented as correlation maps with values ranging from 0 (light) to 1 (dark). There is a clear difference in behavior between the water(Fig2.36a) simulation results and the vacuum (Fig2.36b) simulation results. There is a clear correlation between the eigenvector sets resulting from the water simulation. The vacuum eigenvector sets only show a clear correlation between the first and the 10^{th} nanoseconds.

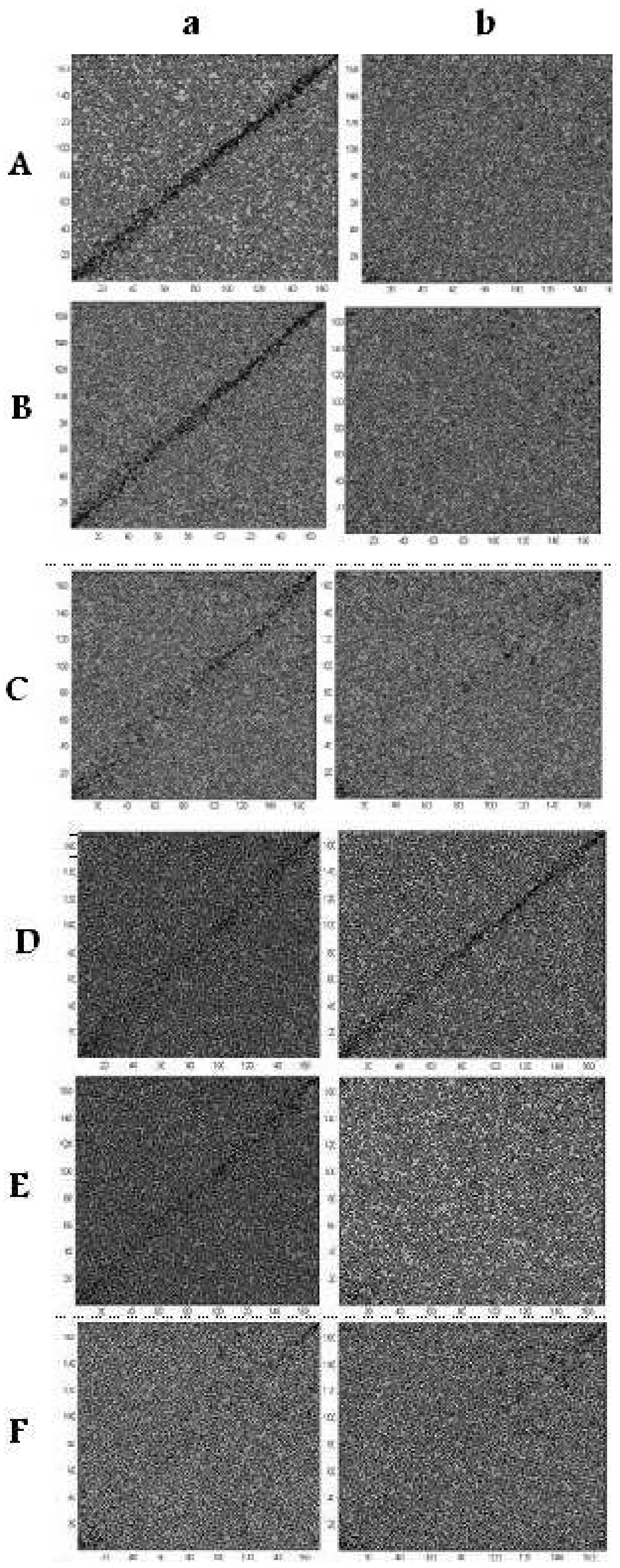


Figure 2.36 Projection of one set of eigenvectors onto another for both water (left column) and vacuum simulation (right column). (A) Eigenvectors calculated from first ns on second ns. (B) Eigenvectors calculated from first ns on third ns. (C) Eigenvectors calculated from first ns on fifth ns. (D) Eigenvectors calculated from first ns on tenth ns. (E) Eigenvectors calculated from first ns on fifteenth ns. (F) Eigenvectors calculated from first ns on twentieth ns.

Chapter 3
Conclusions

A 20 ns molecular dynamics simulation was performed on the protein *BPTI* in water and in vacuum. This is the longest reported simulation in the literature. These simulations were analyzed and compared. The following are the results:

1. The B-factor analysis shows qualitative agreement with experiment, with the vacuum results showing slightly better agreement.

2. The *RMSD* values converged close to 2 angstroms in water simulation and 1.45 angstroms in vacuum simulation.

3. A more dynamic correlation map resulted from the water simulation. This could be the starting point for future research to try and determine which residues correlate exactly with each other.

4. The *PCA* technique results show that only a few eigenvectors, or correlated motions, can describe the significant non-random motion of the protein in water and in vacuum.

5. The water simulation showed clear correlated motion along the significant eigenvectors, while the vacuum simulation did not.

6. The analysis showed that a 1ns simulation was not enough to give repeatable results for motions of the protein along significant eigenvectors.

7. The water simulation showed clear states where it spent time along certain eigenvectors. The vacuum simulation did not.

8. By looking at the probability distribution of the motion amplitude, it was made clear that the only non-random motions were along significant eigenvectors.

9. Projecting sets of eigenvectors from different time regions of the 20 ns simulation show repeatability for the water simulation only.

There are plenty of future research directions that are possible based on this work. The most attractive would be to try and point at specific motions and explain their physical or chemical origin: Which atom pair initiates a certain bending motion, or a certain open-close mechanism? The answers to these questions could be the key to controlling the motions and the function of this and other proteins.

Appendix A
Routine java Code

A.1 rotation class

```
import java.io.*; import java.util.Date; public class dcd {
FileWriter out; PrintWriter outdata;
  public dcd() {
  }
int[] backbone;
  public static void main(String[] args)throws IOException {
    dcd dcd1 = new dcd();
    dcd1.read_dcd_fit();
  }
  /*find the squre of value*/
  public float sqr(float sq)
  {
  return sq*sq;
  }
  /*calculate rmsd between two frame*/
  public double rmsd(Frame ref, Frame  mob)
  {
  double dx,dy,dz,dr;
  dr=0;
  for(int i=0;i<mob.num_Atom;i++)
  {

  dx=sqr( mob.atoms[i].get_X() - ref.atoms[i].get_X());
  //dx=dx*dx;
  dy=sqr(mob.atoms[i].get_Y() - ref.atoms[i].get_Y());
//  dy=dy*dy;
  dz=sqr(mob.atoms[i].get_Z() - ref.atoms[i].get_Z());
 // dz=dz*dz;
  dr+=dx+dy+dz;

  }
  dr=Math.sqrt(dr/mob.num_Atom);
  return (dr);
  }//end rmsd
   public double rmsd(float []com, Frame  mob)
  {
  double dx,dy,dz,dr;
  dr=0;
  for(int i=0;i<mob.num_Atom;i++)
  {

  dx=sqr( mob.atoms[i].get_X() - com[0]);
  //dx=dx*dx;
  dy=sqr(mob.atoms[i].get_Y() - com[1]);
//  dy=dy*dy;
  dz=sqr(mob.atoms[i].get_Z() - com[2]);
 // dz=dz*dz;
  dr+=dx+dy+dz;

  }
  dr=Math.sqrt(dr/mob.num_Atom);
  return (dr);

  }//end rmsd
  public void read_dcd_fit()throws IOException
  {
  int[] backbone={4 ,27 ,42 ,54 ,76 ,84 ,103 ,117 ,131 ,146 ,167 ,181 ,187
   ,204 ,212 ,234 ,244 ,268 ,287 ,306 ,330 ,351
 ,371 ,392 ,406 ,416 ,438 ,448 ,455 ,476 ,484 ,501 ,515 ,535 ,551 ,572 ,579 ,
 588 ,596 ,620 ,630 ,652
 ,676 ,690 ,704 ,724 ,746 ,757 ,767 ,782 ,796 ,804 ,821 ,845 ,861 ,869 ,879};
  out =new FileWriter("c:\\val.txt");
  outdata=new PrintWriter(out);
//FileInputStream readdcd=new FileInputStream("F:\\thesis data\\ca_1-2ns.dcd");
   FileInputStream readdcd=new FileInputStream("C:\\Program Files\
   \University of Illinois\\VMD\\proteins\\alanin.dcd");
  DataInputStream ReadDcdData=new DataInputStream(readdcd);
 // System.out.println( ReadDcdData.readInt());
   int nn = ReadDcdData.readInt();System.out.println(nn);
  ReadDcdData.skipBytes(4);
  int number_frame = ReadDcdData.readInt();
  System.out.println("number of frame: " + number_frame );
  ReadDcdData.skipBytes((nn-8));



  System.out.println();
   System.out.println( ReadDcdData.readInt());
   System.out.println( ReadDcdData.readInt());

  //end first part
  ReadDcdData.skipBytes(164);
   System.out.println( ReadDcdData.readInt());
    System.out.println( ReadDcdData.readInt());
    ReadDcdData.skipBytes(4);
    System.out.println( ReadDcdData.readInt());
    //end secand part
//end part three
    int s= ReadDcdData.readInt();
    System.out.println();
   System.out.println( s/4+" number of atoms");

Frame ref=new Frame(s/4);
 for(int i=0;i<s/4;i++)
  {
  ref.atoms[i].x=  ReadDcdData.readFloat();
  }
ReadDcdData.readInt(); /*Begin for all frame*/ Date end=new
Date();

ReadDcdData.readInt(); for(int i=0;i<s/4;i++)
  {
  ref.atoms[i].y=  ReadDcdData.readFloat();
  }
ReadDcdData.readInt(); ReadDcdData.readInt(); for(int
i=0;i<s/4;i++)
  {
  ref.atoms[i].z=  ReadDcdData.readFloat();
  }
ReadDcdData.readInt();
//end refrence frame
//begin mobile frame
```

```
int kk=1000; System.out.println(end.toString()); Frame mob=new
Frame(s/4);
//for(int ss=0 ;ss< 41300;ss++){
for(int ss=0 ;ss< 100;ss++){
  ReadDcdData.readInt();


for(int i=0;i<s/4;i++)
  {
  mob.atoms[i].x=  ReadDcdData.readFloat();
  }
ReadDcdData.readInt(); ReadDcdData.readInt(); for(int
i=0;i<s/4;i++)
  {
  mob.atoms[i].y=  ReadDcdData.readFloat();
  }
ReadDcdData.readInt(); ReadDcdData.readInt(); for(int
i=0;i<s/4;i++)
  {
  mob.atoms[i].z=  ReadDcdData.readFloat();
  }
ReadDcdData.readInt();
//end mobil



mob=translate(ref,mob);
//mob=translate(mob);
//ref=translate(ref);
float comx[]=new float[3]; comx=center_frame(ref); float
comy[]=new float[3]; comy=center_frame(mob);
//md.fit meme=new md.fit();

/* a) compute R matrix
* R= r(i,j)= sum(w(n)*(y(n,i)-comy(i))*(x(n,j)-comx(j)))
*/ float R[][]=new float[3][3];
//R=meme.matfit(ref,mob,R);
//mob=new_data(R,mob);
//System.out.println(rmsd(ref,mob) +" new algo");
for(int i=0;i<3;i++)
```

```
  {
    for(int j=0;j<3;j++)
    {
    float tmp=0;
    //int nx=0,ny =0,nz=0;
      for(int n = 0 ; n < ref.num_Atom ; n++)
      {
    /*  tmp+=(mob.atoms[n].get_corr(i)-comy[i])*(ref.atoms[n].get_corr(j)-comx[
      /(ref.num_Atom*ref.num_Atom);*/
      tmp+=(mob.atoms[n].get_corr(i))*(ref.atoms[n].get_corr(j));

      }
     R[i][j]=tmp;
    }
  }
  //show rotation matrix
/* for(int i=0;i<3;i++)
  {
    for(int j=0;j<3;j++)
    {System.out.print(R[i][j]+"\t");
    }System.out.println();
  }*/
  /* find Rt R transpose*/
  float Rt[][]=new float[3][3];
  for(int i=0;i<3;i++)
  {
    for(int j=0;j<3;j++)
    {Rt[i][j]=R[j][i];
    }
  }
   /* find RtR transpose*/
  float RtR[][]=new float[3][3];
  RtR=matix_multiplay(Rt,R);
  float tmp_RtR[][]=new float[4][4];
  for(int i=0;i<3;i++)
  {
    for(int j=0;j<3;j++)
    tmp_RtR[i][j]=RtR[i][j];
  }
/* //test
```

```
  tmp_RtR[0][0]=112;tmp_RtR[0][1]=137;tmp_RtR[0][2]=162;
  tmp_RtR[1][0]=129;tmp_RtR[1][1]=159;tmp_RtR[1][2]=189;
  tmp_RtR[2][0]=146;tmp_RtR[2][1]=181;tmp_RtR[2][2]=162;*///end test

  /*b) find the eigenvalues and eigenvectors of RtR by jaccobi method*/
  float evalue[]=new float[3];
  float evector[][]=new float[3][3];
  md.jacobi jacobi1=new md.jacobi();
   float eye[][]=new float[3][4];
  eye=jacobi1.jacobi(tmp_RtR,evalue,evector);

  //System.out.println("eye eye");
  //show e value e vector
   for(int i=0;i<3;i++)
  {
    for(int j=0;j<3;j++)
    {evector[i][j]=eye[i][j];
    }
  evalue[i]=eye[i][3];
  }

  /* transposition the evector matrix to put the vectors in rows*/
  float vectmp;
  vectmp=eye[0][1]; eye[0][1]=eye[1][0]; eye[1][0]=vectmp;
  vectmp=eye[0][2]; eye[0][2]=eye[2][0]; eye[2][0]=vectmp;
  vectmp=eye[2][1]; eye[2][1]=eye[1][2]; eye[1][2]=vectmp;

 /* for(int i=0;i<3;i++)
  {
    for(int j=0;j<4;j++)
    {System.out.print(eye[i][j]+"\t");
    }System.out.println();
  }*/
//  System.out.println("rororoto");
  /* b) 4) sort so that the eigenvalues are from largest to smallest
  *    (or rather so a[0] is eigenvector with largest eigenvalue, ...)*/
  if (eye[0][3] < eye[1][3]) {
    SWAP_Evector(eye,0, 1);
  }
  if (eye[0][3] < eye[2][3]) {
    SWAP_Evector(eye,0, 2);
  }
  if (eye[1][3] < eye[2][3]) {
    SWAP_Evector(eye,1, 2);
  }

  //show eye
 /* for(int i=0;i<3;i++)
  {
    for(int j=0;j<4;j++)
    {System.out.print(eye[i][j]+"\t");
    }System.out.println();
  }*/
  /* c) determine b(i) = R*a(i) or {R*eye}*/
  float b[][]=new float[3][3];
   for(int i=0;i<3;i++)
  {
    for(int j=0;j<3;j++)
    {float tmp=0;
    for(int k=0;k<3;k++)
    {
    tmp+=R[j][k]*eye[i][k];
    }
      b[i][j]=tmp/(float)Math.sqrt((double) eye[i][3]);
      }
  }//end b(i)
 //normalize b(i)
 /*  for(int i=0;i<3;i++)
  {float tmp=0;
    for(int j=0;j<3;j++)
    {
    tmp+=b[i][j]*b[i][j];
    }
    tmp=(float)Math.sqrt((double)tmp);
    for(int j=0;j<3;j++)
    {
    b[i][j]=b[i][j]/tmp;
    }
  }*/
/* d) compute U = u(i,j) = sum(b(k,i) * a(k,j))*/
```

```
 float U[][]=new float[3][3];
   for(int i=0;i<3;i++)
  {
    for(int j=0;j<3;j++)
    {float tmp=0;
    for(int k=0;k<3;k++)
    {
    tmp+=b[i][k]*eye[j][k];
    }
      U[i][j]=tmp;
      }
  }
 //end U
/* System.out.println("rorororo U");
 for(int i=0;i<3;i++)
  {
    for(int j=0;j<3;j++)
    {System.out.print(U[i][j]+"\t");
    }System.out.println();
  }*/
  /*Make allignment between two frames*/
mob=new_data(U,mob);
//comx=center_frame(ref);
//comy=center_frame(mob);
//System.out.println(comx[0]+"\t"+comx[1]+"\t"+comx[2]+" hhhh");
//System.out.println(comy[0]+"\t"+comy[1]+"\t"+comy[2]+" hhjj");
/*open file to write row*/

/* for (int i=0;i< mob.num_Atom;i++) { for (int j=0 ;j<
backbone.length;j++) {
        if (i==backbone[j]) {
outdata.print(mob.atoms[i].get_corr(0)+"\t"+mob.atoms[i].get_corr(1)+"\t"+
mob.atoms[i].get_corr(2)+"\t"); } } } outdata.println();*/


//end write fitted data

Date basem=new Date();;
//end.getTime();
basem.getTime();
```

```
if(ss%kk==999) System.out.println(basem+"\t"+ss);
comx=center_frame(ref);

System.out.println(rmsd(comx,mob));
//System.out.println("end rmsd fit align "+rmsd(moba,mob));

  }//end of for loop for all frame
outdata.close();

  }//end of read dcd file
  //find center of each frame
public float[] center_frame(Frame frame1) { float center[]=new
float[3]; center[0]=0; center[1]=0; center[2]=0; for(int i=0;
i<frame1.num_Atom; i++) { center[0]+=frame1.atoms[i].x;
center[1]+=frame1.atoms[i].y; center[2]+=frame1.atoms[i].z; }//end
for center[0]=center[0]/frame1.num_Atom;
center[1]=center[1]/frame1.num_Atom;
center[2]=center[2]/frame1.num_Atom;

return center; }//end center of frame method public float[][]
matix_multiplay(float A[][],float B[][]) { float C[][]=new
float[3][3]; for(int i=0;i<3;i++)
  {
    for(int j=0;j<3;j++)
    {float tmp=0;
      for(int s=0; s<3 ; s++)
      tmp+= A[i][s] * B[s][j];
    C[i][j]=tmp;
    }
   }
return C; }//end matrix multiplay /* function swap the col
transform to rows */ public float[][] SWAP_Evector(float
vector[][], int i, int j ) {
     float v,v1;
     for(int s=0; s<4;s++)
     {
    v = vector[i][s]; vector[i][s] = vector[j][s]; vector[j][s] = v;
      }

      return vector;
```

```
} /* find the new dat after the fit and align*/ public Frame
new_data(float U[][], Frame frame) { for(int
kk=0;kk<frame.num_Atom;kk++) { for(int i=0;i<3;i++)
  {float tmp=0;
    for(int j=0;j<3;j++)
    {
         tmp+=U[i][j]*frame.atoms[kk].get_corr(j);
    }frame.atoms[kk].set_corr(i,tmp);

  }
 // System.out.println(frame.atoms[kk].get_corr(0)
 // +"\t"+frame.atoms[kk].get_corr(1)+"\t"+frame.atoms[kk].get_corr(2));
} return frame; }//end method /*translation removal vector*/
public Frame translate(Frame ref) { float Rcen[]=new float[3];
float origin[]=new float[3]; float Dist[]=new float[3]; Rcen=
center_frame(ref); origin[0]=0;origin[1]=0;origin[2]=0; for(int
i=0; i<3;i++) Dist[i]= Rcen[i]-origin[i]; for(int i=0;
i<ref.num_Atom;i++) { for(int s=0;s<3;s++)
ref.atoms[i].set_corr(s,ref.atoms[i].get_corr(s)-Dist[s]);

} Rcen= center_frame(ref);
//System.out.println(Rcen[0]+"\t"+Rcen[1]+"\t"+Rcen[2]);
return ref;

} public Frame translate(Frame ref1,Frame mob1) { float Rcen[]=new
float[3]; float Mcen[]=new float[3]; float Dist[]=new float[3];
Rcen= center_frame(ref1); Mcen= center_frame(mob1); for(int i=0;
i<3;i++) Dist[i]=Mcen[i]- Rcen[i];

for(int i=0; i<mob1.num_Atom;i++) { for(int s=0;s<3;s++)
mob1.atoms[i].set_corr(s,mob1.atoms[i].get_corr(s)-Dist[s]);

} return mob1; }

}//end class
```

A.2 Corrolation and Analysis class

```
import java.io.*; public class anadcd {
 float A[][]=new float[20000][171];
```

```
  public anadcd() {
  }
  public static void main(String[] args)throws IOException {
    anadcd anadcd1 = new anadcd();
    float sum[]=new float[171];
    float ava[]=new float[171];
    for (int m=0;m<171;m++) sum[m]=0;
    int N_raw=20000;
    int N_col=171;
    float Cov[][]=new float[N_col][N_col];
    for(int i=0; i<N_col; i++)
        for (int j=0; j<N_col; j++)
          Cov[i][j]=0;
    FileInputStream readdcd;
    DataInputStream ReadDcdData;
/*loop until the end of data  */
//for (int nano=1;nano<21;nano++){
/*loop to find colom avarages*/ for (int ns=20;ns<21;ns++){
  readdcd=new FileInputStream("D:\\Documents and Settings\\Administrator\\allva
  //readdcd=new FileInputStream("D:\\newdata\\20-fa-nsv.dcd");
  //readdcd=new FileInputStream("D:\\basem\\"+ns+"ns");
  ReadDcdData=new DataInputStream(readdcd);
 /* System.out.println(ReadDcdData.readInt());
  ReadDcdData.skipBytes(88);
  ReadDcdData.skipBytes(172);*/
  ReadDcdData.skipBytes(276);
  ReadDcdData.skipBytes(3540000);
for (int m=0;m<171;m++) sum[m]=0; for(int i=0; i<N_col; i++)
        for (int j=0; j<N_col; j++)
          Cov[i][j]=0;
//int m= ns-1;
for (int m=0;m<ns;m++){
//ReadDcdData.skipBytes(20000*236*m);
//end part three
int s1=228; int fra =0; System.out.println( s1/4+" number of atoms
"+ ns);

while(fra<20000)/*begin get data all 20000 frame from file*/ {
for(int j=0;j<3;j++)//for begin get data frame from file {
    ReadDcdData.readInt();
    for(int i=0;i<57;i++)
    anadcd1.A[fra][3*i+j]=ReadDcdData.readFloat();
    ReadDcdData.readInt();
}//end for 1 end hold all frame data fra++; }//end while for all
20000 frame for(int j=0;j<N_col;j++)
   sum[j]+=anadcd1.Sum_coloun(j,N_raw);/* add all values*/
}//end for m
 ReadDcdData.close();
 readdcd.close();
/*find over all average*/
 for(int j=0;j<N_col;j++)
  {  ava[j]=anadcd1.ava_Col_Number(sum[j],N_raw,ns);
 // System.out.println(ava[j]);
  }
 FileWriter avadata=new FileWriter("d:\\newdata\\vacuum\\vava-"+ns+".txt");
   for(int j=0;j<171;j++)
        {
      avadata.write(ava[j]+"\n");
        }
  avadata.close();
/*find the covariance matrix for each nanosecand then add them*/
  readdcd=new FileInputStream("D:\\Documents and Settings\\Administrator\\allva
  // readdcd=new FileInputStream("D:\\newdata\\20-fa-nsv.dcd");
  //readdcd=new FileInputStream("D:\\basem\\"+ns+"ns");
  ReadDcdData=new DataInputStream(readdcd);
 /* FileOutputStream writedcd=new FileOutputStream("d:\\newdata\\ava"+ns+".dcd"
  DataOutputStream WriteDcdData=new DataOutputStream(writedcd);
 /* System.out.println(ReadDcdData.readInt());
  ReadDcdData.skipBytes(88);
  ReadDcdData.skipBytes(172);
  ReadDcdData.skipBytes(12);//end part three*/
  byte store[]=new byte[276];
  ReadDcdData.read(store);
   ReadDcdData.skipBytes(3540000);
 // WriteDcdData.write(store);//write the header for the dcd file
//ReadDcdData.skipBytes(20000*236*m);
  for (int m=0;m<ns;m++){
 int s1=228;
 int fra =0;
System.out.println("atoms "+ m);
```

```
while(fra<20000) { for(int j=0;j<3;j++)//for 1 {
    ReadDcdData.readInt();
    for(int i=0;i<57;i++)
    anadcd1.A[fra][3*i+j]=ReadDcdData.readFloat();
    ReadDcdData.readInt();
}//end for 1 fra++; }//end while

for(int j=0;j<N_col;j++)
   {
    for(int i=0;i<N_raw;i++)
      {
      anadcd1.A[i][j]=anadcd1.Div_from_Ava(anadcd1.A[i][j],ava[j]);
      }
   }
   /*Write the average to the dcd file*//*
fra =0; while(fra<20000) { for(int j=0;j<3;j++)//for 1 {
    WriteDcdData.writeInt(57*4);
    for(int i=0;i<57;i++)
    WriteDcdData.writeFloat(anadcd1.A[fra][3*i+j]);
    WriteDcdData.writeInt(57*4);
}//end for 1 fra++; }//end while /*the new covariance matrix is
look like*/

   for(int i=0;i<N_col;i++)
      {
      for(int j=0;j<=i;j++)
        {
        Cov[i][j]+=anadcd1.Dot_Col(i,j,N_raw);//here rasha
        Cov[j][i]=Cov[i][j];
        }
       }

}//end for m/*if i need to do*/ for(int i=0;i<N_col;i++)
      {
      for(int j=0;j<=i;j++)
        {
          Cov[i][j]=Cov[i][j]/(N_raw*(ns));
          Cov[j][i]=Cov[i][j];
        }
      }
     /* for(int j=0;j<N_col;j++)
      {
        for (int i = 0; i <= j; i++)
        {
          Cov[j][i] = Cov[i][j];
        }
      }*/

 ReadDcdData.close();
 readdcd.close();
 //WriteDcdData.close();
/*find Bfactor and save them in file */
   float bfactor[]=new float[57];
   bfactor=anadcd1.Bfactor(Cov);
   FileWriter Bfactorf=new FileWriter("d:\\newdata\\vacuum\\vBfactort_"+ns+".tx
   for(int j=0;j<57;j++)
        {
      Bfactorf.write(bfactor[j]+"\n");
        }
   Bfactorf.close();
       FileWriter covf=new FileWriter("d:\\newdata\\vacuum\\"+ns+"-covv.txt");
       //show covaraince matrix in file
      for(int i=0;i<N_col;i++)
      {
      for(int j=0;j<N_col;j++)
        {
      covf.write(Cov[i][j]+"\t");
        }covf.write("\n");
      }
      covf.close();

      System.out.println("color matrix");
  /*    try{
     // open file to writ matrix of color number
     // plot=new FileOutputStream("c:\\plot.txt");
     FileWriter plots1 =new FileWriter("c:\\newdata\\plot_1-ns.txt");
      for(int i=0;i<N_col;i++)
      {
      for(int j=0;j<N_col;j++)
```

```
         {
         for(int s=1;s<Range.length;s++)
          {
          if ((Cov[i][j] >= Range[s-1]) || (Cov[i][j] > Range[s]))
        plots1.write(s);
          }plots1.write("\n");
          //plots.print(analysis1.color_matrix[198*i+j]);
         // System.out.print(analysis1.color_matrix[198*i+j]+",");
         }
        // plots.println();
       }plots1.close();
       }catch (IOException ex) {System.out.println(ex.getMessage());}
*/

  }//end ns
//}//endnano
  }//end main
 /*find the sum of all col*/
  public  float Sum_coloun( int Y_colNumber,int N_Raw)
  {float sum1;

  sum1=0;
  for(int i=0;i<N_Raw;i++)
  sum1+=A[i][Y_colNumber];
  return sum1;
  }
   //find the avarage fro each col
  public float ava_Col_Number(float Sum_Col_Num,float Num_Raw,int ns)
  {
  return Sum_Col_Num/(Num_Raw*ns);
  }
  //find divaction from avarage
  public float Div_from_Ava(float value,float avarage)
  {
  return value-avarage;
  }
  //multiplay two col
  public float Dot_Col(int i_col1,int j_col2,int Raw_length)
  {
  float val=0;
  for(int s=0;s<Raw_length;s++)
  {
  val+=A[s][i_col1]*A[s][j_col2];
  }
  return val;
  }
  public float absval(float val)
  {if (val >= 0) return val;
   else return -val;
  }
  public float min(float a,float b)
  {
  if(a>b) return b;
  else return a;
  }
  public float max(float a,float b)
  {
  if(a>b) return a;
  else return b;
  }
/*calculate the Bfactor (tempreture factor)*/ public float[]
Bfactor(float [][]cov) {float bfactor[]=new float[57];

for(int i = 0; i < 57; i++)
  {float sum=0;
  for(int j = 0; j < 3; j++)
  sum+=cov[3 * i + j][3 * i + j];
  bfactor[i] = (8/3)*(22/7)*(22/7)*(sum);
  }
  return bfactor;
} }
```

A.3 projection class

```
import java.io.*; public class eigproject {
  public eigproject() {
  }
  public static void main(String[] args)throws IOException {
    eigproject eigproject1 = new eigproject();
    float eign1[][] = new float[171][171];
    float eign2[][] = new float[171][171];
    int nano1 = 1;//nano1<11;naon1++){
    FileReader read = new FileReader("d:\\newdata\\pca_gw\\eigvd"+ nano1 + ".tx
      StreamTokenizer in = new StreamTokenizer(read);
      for (int i = 0; i < 171; i++){
        for (int j = 0; j < 171; j++) {
          in.nextToken();
          if (in.ttype == StreamTokenizer.TT_NUMBER){
            eign1[i][j] = (float)in.nval;
            //System.out.print(eign1[i][j]+"\t");
          }
        }//System.out.println();
      }/*output file to write data*/
      int nano2 = 20;
   FileReader read2 = new FileReader("d:\\newdata\\pca_gw\\eigvd"+ nano2 + ".tx
     StreamTokenizer in2 = new StreamTokenizer(read2);
     for (int i = 0; i < 171; i++){
       for (int j = 0; j < 171; j++) {
         in2.nextToken();
         if (in2.ttype == StreamTokenizer.TT_NUMBER){
           eign2[i][j] = (float)in2.nval;
           //System.out.print(eign2[i][j]+"\t");
         }
       }//System.out.println();
     }/*output file to write data*/
FileWriter wr = new FileWriter("D:\\newdata\\pca_gw\\map120.txt");
    /*project just two eigvector to make thier plane*/
    for (int i = 0; i < 171; i++)
    {
       for (int j = 0; j < 171; j++)
       {float sum=0;
         for (int s = 0; s < 171; s++)
         {
           sum+=eign1[s][i]*eign2[s][j];
         }//System.out.println(sum);
         wr.write(sum+" \t");
       }wr.write(" \n");
    }wr.close();
       }
  }
```

A.4 Extract cα from Trajectory DCD

```
import java.io.*; public class getcalphacoor {
  public getcalphacoor() {
  }
  public static void main(String[] args) throws IOException{
  getcalphacoor getcalphacoor1 = new getcalphacoor();
  int caindex[]={4, 27, 42, 54, 76, 84, 103, 117, 131, 146, 167, 181, 187, 204,
      212, 234, 244, 268, 287, 306 ,330, 351, 371, 392, 406 ,416, 438, 448 ,455
      ,476 ,484 ,501 ,515 ,535 ,551 ,572 ,579 ,588, 596, 620, 630, 652 ,676
      ,690 ,704 ,724 ,746 ,757 ,767 ,782 ,796 ,804 ,821 ,845 ,861 ,869 ,879};
  float A[] = new float[1000];
  float CaCoorFrame[]=new float[caindex.length];
  FileInputStream readdcd=new FileInputStream("d:\\catdcd\\20nsv.dcd");
  DataInputStream ReadDcdData=new DataInputStream(readdcd);
  FileOutputStream writedcd = new FileOutputStream("D:\\ca2.dcd");
  DataOutputStream WriteDcdData = new DataOutputStream(writedcd);
  byte store[] = new byte[276];
  float copy[]=new float[882];
  float index[]={0,0,0,0,0};
  ReadDcdData.read(store);
  WriteDcdData.write(store);
  ReadDcdData.close();readdcd.close();
  readdcd=new FileInputStream("z:\\vacuum-assgin.dcd");
  ReadDcdData=new DataInputStream(readdcd);
  ReadDcdData.skipBytes(312);
  //for (int k = 0; k < 316; k++)System.out.print((char)store[k]);
  int x=0;
  System.out.println(caindex.length);
  while(x<400200)
  {
  ReadDcdData.readInt();

  for (int kk = 0; kk < 882; kk++) {/* {for (int k = 0; k < 4; k++)
                       index[k]= ReadDcdData.readByte();
                       index[4]=index[0]+index[1]+index[2]+index[3];
                       System.out.println(index[4]+"\t"+kk);
*/copy[kk]= ReadDcdData.readFloat();
                       }
  ReadDcdData.readInt();
 // System.out.println(copy[4]+"\t");
  WriteDcdData.writeInt(57*4);
  for (int k = 0; k < 57; k++)
  {
      WriteDcdData.writeFloat(copy[caindex[k]-1]);
  }
  WriteDcdData.writeInt(57*4);
  x++;
  if (x%10000==0) System.out.println(x);
  }
  readdcd.close();
  }

}
```

A.5 Convert DCD File Format To PDB format class

```
import java.io.*; public class producexyzdata {
  public producexyzdata() {
  }
  public static void main(String[] args) throws IOException{
    producexyzdata producexyzdata1 = new producexyzdata();
    float A[]=new float[171];
    for(int ns=9;ns<21;ns++){
      FileInputStream readdcd = new FileInputStream("D:\\Documents and Settings
    // FileInputStream readdcd = new FileInputStream("D:\\basem\\"+ns+"ns");
      DataInputStream ReadDcdData = new DataInputStream(readdcd);
      FileWriter wr = new FileWriter("D:\\v"+ns+"ns.txt");
      System.out.println(ReadDcdData.readInt());
      ReadDcdData.skipBytes(88);
      ReadDcdData.skipBytes(172);
      ReadDcdData.skipBytes(12);
      /*read each fame and write it in out file*/
      int s1=228;
int fra =0; System.out.println( s1/4+" number of atoms "+ ns);
while(fra<20000)/*begin get data all 20000 frame from file*/ {
for(int j=0;j<3;j++)//for begin get data frame from file {
    ReadDcdData.readInt();
    for(int i=0;i<57;i++)
    A[3*i+j]=ReadDcdData.readFloat();
    ReadDcdData.readInt();
}//end for 1 end hold all frame data fra++; for(int i=0;i<171;i++)
wr.write(A[i]+"\t");
    wr.write(" \n");
}//end while for all 20000 frame wr.close();
  ReadDcdData.close();
  readdcd.close();
    }//end each ns
  }

}
```

A.6 Extract Important Data From NAMD Logfile class

```
import java.io.*; public class Namdlog {

  public Namdlog() {

  }
  public static void main(String[] args) throws IOException{
    Namdlog namdout1 = new Namdlog();
    FileReader fr=new FileReader("D:\\basimout.txt");
    boolean app=true;
    FileWriter fw=new FileWriter("D:\\matlabtest\\vacuumENERGY1.txt",app);
    StreamTokenizer in=new StreamTokenizer(fr);
    in.nextToken();
    String remark;
    int counttitels=0;
    int countenergy=0;
    boolean energytiming=true;
    String ETITEL[]=new String[15];
    double ENERGY[]=new double[15];
     double ENERGY2 =100;
   double ENERGY1 =10;
    while (in.ttype != StreamTokenizer.TT_EOF)
  {
```

```
if (in.ttype == StreamTokenizer.TT_WORD)//first if
{
remark=in.sval;


if (remark.equals("ETITLE")) //2ed if
  {
   counttitels++;
      if (counttitels==1) //3ed if
      {

       for(int i=0;i<15;i++)
         if (in.nextToken() == StreamTokenizer.TT_WORD) ETITEL[i]=in.sval;
         for(int i=1;i<15;i++){System.out.print(ETITEL[i]+"\t");}
         //System.out.println();
      }//3ed if
   }//2ed if
} //first if
//end read header
//begin read energy data
if (in.ttype == StreamTokenizer.TT_WORD)//first if
{remark=in.sval;

if (remark.equals("ENERGY")) //2ed if
  {
  countenergy++;

      if ((countenergy!=1)&& (countenergy!=0))//3ed if
      {
        for(int i=0;i<15;i++)
        if (in.nextToken() == StreamTokenizer.TT_NUMBER)
            {ENERGY[i]=in.nval;}
        if (ENERGY[1]%1000!=900 )
        {if ((ENERGY[12]>=(ENERGY1-30))&&(ENERGY[12]<=(ENERGY1+30)))
        // if ((ENERGY[10]>=(ENERGY2-100))&&(ENERGY[10]<=(ENERGY2+100)))
        {
        fw.write(ENERGY[1]+"\t"+ENERGY[12]+"\t"+ENERGY[10]+"\t");

        }ENERGY1=ENERGY[12];ENERGY2=ENERGY[10];
```

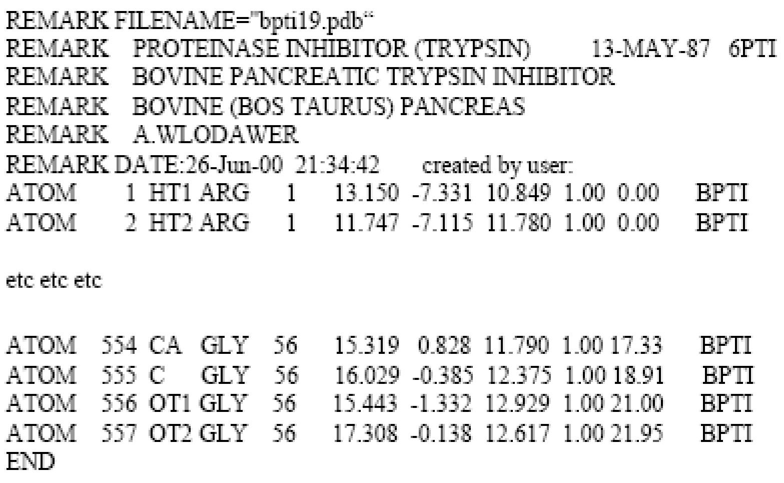
```
            }
            fw.write(" \n");

          }//3ed if
      }//2ed if
    } //first if
   in.nextToken();
   }//end while
fw.close();
  }

}
```

Appendix B
Files Format

B.1 *PDB* Format

PDB Files The term PDB can refer to the Protein Data Bank (http://www.rcsb.org/pdb/), to a data file provided there, or to any file following the PDB format. Files in the PDB include information such as the name of the compound, the species and tissue from which is was obtained, authorship, revision history, journal citation, references, amino acid sequence, stoichiometry, secondary structure locations, crystal lattice and symmetry group, and finally the ATOM and HETATM records containing the coordinates of the protein and any waters, ions, or other heterogeneous atoms in the crystal. Some PDB files include multiple sets of coordinates for some or all atoms. Due to the limits of x-ray crystallography and NMR structure analysis, the coordinates of hydrogen atoms are not included in the PDB.

Here are the ATOM records for the first two residues of ubiquitin from the 1UBQ entry in the PDB:



```
REMARK FILENAME="bpti19.pdb"
REMARK   PROTEINASE INHIBITOR (TRYPSIN)        13-MAY-87   6PTI
REMARK   BOVINE PANCREATIC TRYPSIN INHIBITOR
REMARK   BOVINE (BOS TAURUS) PANCREAS
REMARK   A.WLODAWER
REMARK DATE:26-Jun-00  21:34:42       created by user:
ATOM      1  HT1 ARG    1     13.150  -7.331  10.849  1.00  0.00     BPTI
ATOM      2  HT2 ARG    1     11.747  -7.115  11.780  1.00  0.00     BPTI

etc etc etc

ATOM    554  CA   GLY   56    15.319   0.828  11.790  1.00 17.33     BPTI
ATOM    555  C    GLY   56    16.029  -0.385  12.375  1.00 18.91     BPTI
ATOM    556  OT1 GLY   56    15.443  -1.332  12.929  1.00 21.00     BPTI
ATOM    557  OT2 GLY   56    17.308  -0.138  12.617  1.00 21.95     BPTI
END
```

Figure B.1 The Protein Data Bank

The fields seen B.1 in order from left to right are the record type, atom ID, atom name, residue name, residue ID, x, y, and z coordinates, occupancy, temperature factor (called beta), segment name, and line number.

B.2 *PSF* Format

B.3 A force field parameter Format

A PSF file, also called a protein structure file, contains all of the molecule-specific information needed to apply a particular force field to a molecular system. The CHARMM force field is divided into a topology file, which is needed to generate the PSF file, and a parameter file, which supplies specific numerical values for the generic CHARMM potential function. The topology file defines the atom types used in the force field; the atom names, types, bonds, and partial charges of each residue type; and any patches necessary to link or otherwise mutate these basic residues. The parameter file provides a mapping between bonded and nonbonded interactions involving the various combinations of atom types found in the topology file and specific spring constants and similar parameters for all of the bond, angle, dihedral, improper, and van der Waals terms in the CHARMM potential function.

The PSF file contains five main sections of interest: atoms, bonds, angles, dihedrals, and impropers (dihedral force terms used to maintain planarity). The following is taken from a PSF file for BPTI. First is the title and atom records: PSF

6 !NTITLE
REMARKS FILENAME="bpti19.psf"
REMARKS PROTEINASE INHIBITOR (TRYPSIN) 13-MAY-87 6PTI
REMARKS BOVINE PANCREATIC TRYPSIN INHIBITOR
REMARKS BOVINE (BOS TAURUS) PANCREAS
REMARKS A.WLODAWER
REMARKS DATE:26-Jun-00 21:34:43 created by user:

557 !NATOM
1 BPTI 1 ARG HT1 HC 0.350000 1.00800 0
2 BPTI 1 ARG HT2 HC 0.350000 1.00800 0
3 BPTI 1 ARG N NH3 -0.300000 14.0067 0
⋮
571 !NBOND: bonds
3 5 5 18 18 19 5 6

```
6 7 7 8 8 9 9 10
⋮
818 !NTHETA: angles
3 5 18 3 5 6 5 18 19
18 5 6 5 6 7 6 7 8
⋮
345 !NPHI: dihedrals
3 5 6 7 5 6 7 8
6 7 8 9 7 8 9 11
⋮
254 !NIMPHI: impropers
5 3 18 6 9 8 11 10
11 12 15 9 22 20 25 23
⋮
112 !NDON: donors
9 10 12 13 12 14 15 16
15 17 3 1 3 2 3 4
⋮77 !NACC: acceptors
19 18 26 25 32 31 33 31
35 34 47 46 54 53 63 62
⋮
24 !NNB
45 44 43 97 96 95 210 209
208 224 223 222 236 235 234 328
⋮
217 0 !NGRP
0 0 0 5 0 0 7 0 0
11 0 0 14 0 0 17 0 0
```

B.4 A configuration file Format

Visit this site

http://www.ks.uiuc.edu/Training/SumSchool/materials/sources/ tutorials/02-namd-tutorial/namd-tutorial-html/node24.html

B.5 Parameter file Format

Visit this site

http://www.ks.uiuc.edu/Training/SumSchool/materials/sources/ tutorials/02-

namd-tutorial/namd-tutorial-html/node23.html

B.6 Topology file Format

Visit this site
http://www.ks.uiuc.edu/Training/SumSchool/materials/sources/ tutorials/02-namd-tutorial/namd-tutorial-html/node22.html

B.7 DCD trajectory format

```
The trajectory DCD format is structured as follows (FORTRAN
UNFORMATTED, with Fortran data type descriptions):

 HDR     NSET ISTRT   NSAVC   5-ZEROS NATOM-NFREAT    DELTA   9-ZEROS

'CORD'  #files  step 1  step    zeroes  (zero)          timestep
(zeroes)
                        interval

C*4     INT     INT     INT     5INT    INT             DOUBLE
9INT

==========================================================================
NTITLE          TITLE

INT (=2)        C*MAXTITL
                (=32)

==========================================================================
NATOM #atoms INT
==========================================================================
X(I), I=1,NATOM         (DOUBLE) Y(I), I=1,NATOM Z(I), I=1,NATOM
==========================================================================
```

Appendix C
NAMD Configration File

C.1 *NAMD* Configration File in Vacuum

```
NAMD configuration file for BPTI

# molecular system

structure bpti.psf

# force field
paraTypeCharmm on

parameters par_all22_prot.inp

exclude     scaled1-4

1-4scaling 0.4

# approximations

switching on

switchdist  8

cutoff 12

pairlistdist 13.5

margin 0

stepspercycle 20

# integrator

rigidbonds  all

timestep 1
```

```
commotion no

# output

outputenergies 50

outputtiming 50

outputpressure    50

outputMomenta     50

binaryrestart no

restartname bpti_bas

restartfreq 50

dcdfile vacuum-sim-test

# run-specific parameters
# molecular system

coordinates bpti_equil.coor

velocities bpti_equil.vel

#output

outputname bpti_sim

numsteps 20000000
```

C.2 *NAMD* Configration File in water

```
#namd configration file for minimize the water molecle while bpti
atom fixed

#molecular system
```

```
structure
/home/basim/namd_2.5b1_linux-i686/thesis/water/raw_data/water_bpti.psf

coordinates
/home/basim/namd_2.5b1_linux-i686/thesis/water/minmization_file/water_bpti_min.

temperature 0



#the full path is needed

#force field

paraTypeCharmm on

parameters toppar/par_all22_prot.inp

parameters toppar/par_all27_prot_lipid.inp

restartname water_bpti.pdb

restartfreq 50

binaryrestart no



#output

outputenergies 50

outputtiming 50

outputMomenta 50

outputPressure 50

xstFreq     50
```

```
dcdFreq     50

wrapAll     on

wrapNearest on


#integrator

#rigidbonds all


timestep 1

nonbondedFreq 2

stepspercycle 20

fullElectFrequency 2 # PME only every other step




#Approximations

switching on

switchdist 8.5

cutoff 10

pairlistdist 11.5
```

```
cellBasisVector1 44.484 0 0

cellBasisVector2 0 40.844 0

cellBasisVector3 0 0 41.317

cellOrigin 7.652 4.835 3.977


margin 5


Pme     on

PmeGridsizeX    32

PmeGridsizeY    32

PmeGridsizeZ    64


#basic simulation

exclude scaled1-4

1-4scaling     0.4


#fix bpti atoms

fixedatoms on

fixedAtomsForces on
```

```
#bpti file fixed add water

fixedatomsfile
/home/basim/namd_2.5b1_linux-i686/thesis/water/raw_data/fix.pdb

fixedAtomsCol B



langevin on

langevinDamping 10

langevinTemp 300

langevinHydrogen no



langevinPiston      on

langevinPistonTarget    1.01325

langevinPistonPeriod    200

langevinPistonDecay 100

langevinPistonTemp  300



useGroupPressure    yes # smaller fluctuations

useFlexibleCell     yes # allow dimensions to fluctuate
independently

useConstantRatio    yes # fix shape in x-y plane
```

```
#output

binaryoutput off
outputname  equil_out_wat_fix

# run one step to get into scripting mode
minimize 0

# turn off until later
langevinPiston  off

# minimize nonbackbone atoms (atom fixed)
minimize 5000
output min_wat_fix

# heat fixed atom (water)
# langevin  on
run 5000
output heat_fix
```

```
#equilibration 10ps

langevinPiston  on

run 10000

output equil_fix


# min all atoms

langevinPiston  off

fixedAtoms  off

minimize 10000

output min_all_wat


# heat all

# langevin  on


run 10000

output heat_all_wat


#equilibration 40ps all

langevinPiston  on

run 40000
```

```
output equil_all_wat



langevinPiston  off

langevin    off


run 20000000

output endsim
```

C.3 PBC Periodic Boundary Condition

Periodic boundary conditions involve surroundings the system (Protein) under study with identical virtual unit cells. the surrounding virtual systems interact with atoms in the real system, creating an environment in which the system effectively sees no vacuum as in (FigC.1). These modeling conditions are effective in eliminating surface interaction of the water

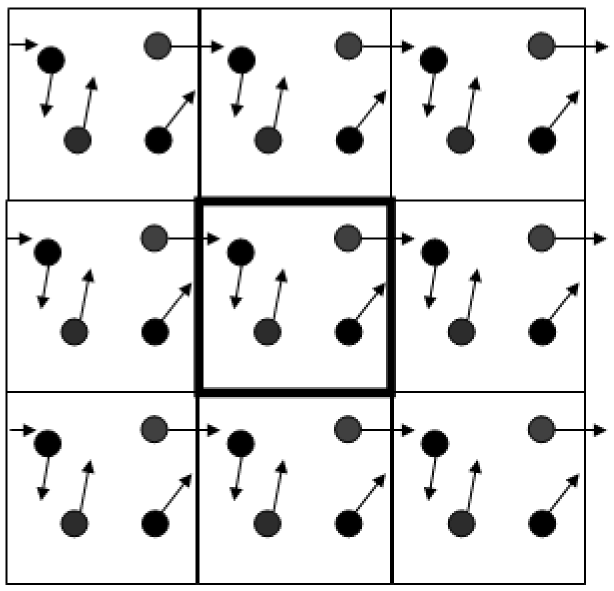


Figure C.1 In Periodic boundary condition when a particle moves in the central box, its periodic image in every one of the other boxes moves with exactly the same orientation in exactly the same way.

molecules which make the system somewhat different than an in vacuum environment.

As one molecule leaves the central box its image enters through the opposite face. No walls at boundary and the system has no surface.

References

1. Saarela, J.A., Tuppurainen, K., Per?kyl?, M., Santa, H. and Laatikainen, R. ”Correlative Motions and Memory Effects in Molecular Dynamics Simulations of Molecules: Principal Components and Rescaled Range Analysis Suggest That the Motions of Native BPTI Are More Correlated Than Those of Its Mutants.” Biophys. Chem., 2002, 95, 49-57.

2. Van Aalten, A comparison of techniques for calculating protein Essential Dynamics, 1996.

3. Huitema, H. Van Liere, R, Interactive Visualization of protein dynamics, IEEE Visualization,465-468 (1999).

4. M. F. van Aalten, J. B. C. Findlay, A. Amadei and H. J. C. Berendsen. Essential dynamics of the cellular Retinol-binding protein: evidence for ligand induced conformational changes. Protein Eng. 8 1129-1136 (1995).

5. G. Chillemi, M. Falconi, A. Amadei, G. Zimatore, A. Desideri and A. Di Nola. The essential dynamics of Cu, Zn superoxide dismutase: suggestion of intersubunit ommunication. Biophys. J., 73, 1007-1018 (1997).

6. A. Amadei, M.A. Ceruso and A. Di Nola. On the convergence of the coordinates basis set obtained by the essential dynamics of proteins molecular dynamics simulations. Proteins: Structure, Function, and Genetics., 36 , 419-424 (1999).

7. I. Daidone, A. Amadei, D. Roccatano and A. Di Nola Molecular dynamics simulation of protein folding by essential dynamics sampling: folding landscape of horse heart cytochrome c. Biophys. J. 85, 2865-2871 (2003).

8. Arcangeli C, Bizzarri AR, Cannistraro S. Molecular dynamics simulation and essential dynamics study of mutated plastocyanin: structural, dynamical and functional effects of a disulfide bridge insertion at the protein surface. Biophys Chem. 2001 Sep 18;92(3):183-99.

9. P. Cioni, E. d. Waal, G. W. Canters, and G. B. Strambini Effects of Cavity-Forming Mutations on the Internal Dynamics of Azurin Biophys. J., February 1, 2004; 86(2): 1149 - 1159.

10. Huber, R. Bennett, Jr., W. S. (1983). Functional significance of flexibility in proteins. Biopolymers, 22, 261-279.

11. Thomas H. Rod, Jennifer L. Radkiewicz, and Charles L. Brooks, III Correlated motion and the effect of distal mutations in dihydrofolate reductase PNAS 100: 6980-6985; published online before print as 10.1073/pnas.1230801100

12. Vitkup D, Ringe D, Karplus M, Petsko GA. (2002) Why protein R-factors are so large: a self-consistent analysis. Proteins. 46:345-354.

13. National Institutes of Health (NIH), Center for Molecular Modeling (CMM) (NIHInfo@OD.NIH.GOV), Molecular Mechanics and Modeling- Why. [Online]Available, November, 2004. http://cmm.info.nih.gov/modeling/guide_documents/molecular _mechanics_document.html.

14. Alder, B. J., and T. E. Wainwright. 1957. Phase transition for a hard sphere system. J. Chem. Phys. 27:1208-1209.

15. Doucet, Jean-Pierre and Jacques,Webber. Computer-Aided Molecular Design. Academic press inc, U.S. San Diego, CA 92101, p123, p171, p127, p138.

16. D.Frenkel and B.Smit, "Understanding Molecular Simulation - From algorithms to applications", Academic Press, 1996.

17. Furio Ercolessi. Historical notes. [Online]Available http://www.fisica.uniud.it/ ercolessi/md/md/node7.html, November, 2004.

18. D.McQuarrie, Statistical Mechanics (Harper Row, New York, 1976), pp. 544-553.

19. F.C.Bernstein, T.F.Koetzle, G.J.Williams, E.E.Meyer Jr., M.D.Brice, J.R.Rodgers, O.Kennard, T.Shimanouchi, M.Tasumi, "The Protein Data Bank: A Computer-based Archival File For Macromolecular Structures," J. of. Mol. Biol., 112 (1977): 535.

20. Laxmikant Kal, Robert Skeel, Milind Bhandarkar, Robert Brunner, Attila Gursoy, Neal Krawetz, James Phillips, Aritomo Shinozaki, Krishnan Varadarajan,

and Klaus Schulten. NAMD2: Greater scalability for parallel molecular dynamics. Journal of Computational Physics, 151:283-312, 1999. http://www.ks.uiuc.edu/Training/SumSchool/materials/sources/tutorials/02-namd-tutorial/namd-tutorial-html/node4.html

21. B.R. Brooks, R.E. Bruccoleri, B.D. Olafson, D.J. States, S. Swaminathan, and M. Karplus, "CHARMM: A Program for Macromolecular Energy, Minimization and Dynamics Calculations," Journal of Computational Chemistry 187(4)(1983).

22. National Institutes of Health (NIH), Center for Molecular Modeling (CMM)
(NIHInfo@OD.NIH.GOV), Molecular Mechanics, [Online]Available, http://cmm.info.nih.gov/modeling/guide_documents/molecular _mechanics_document.html, November, 2004.

23. Laxmikant Kal, Robert Skeel, Milind Bhandarkar, Robert Brunner, Attila Gursoy, Neal Krawetz, James Phillips, Aritomo Shinozaki, Krishnan Varadarajan, and Klaus Schulten. (Guide) NAMD2: Greater scalability for parallel molecular dynamics. Journal of Computational Physics, 151:283-312, 1999. http://www.ks.uiuc.edu/Research/namd/current/ug/

24. Dimassi, M., Sjöstrand, J.: Spectral Asymptotics in the Semiclassical Limit, London Math. Soc. Lecture Notes Series 268, Cambridge University Press (1999).

25. Kim Baldridge. Molecular Mechanics. [Online]Available http://www.sdsc.edu/ kimb/molmech.html, November, 2004.

26. Kian-Tat Lim, Sharon Brunett, Mihail Iotov, Richard B. McClurg, Nagarajan Vaidehi, Siddharth Dasgupta, Stephen Taylor, William A. Goddard III: Molecular dynamics for very large systems on massively parallel computers: The MPSim program. Journal of Computational Chemistry 18(4): 501521 (1997) K_T_ Lim Thesis Chapter 2.

27. B. R. Brooks, R. E. Bruccoleri, B. D. Olafson, D. J. States, S. Swaminathan, and M. Karplus, CHARMM: A Program for Macromolecular Energy, Minimization, and Dynamics Calculations, J. Comp. Chem. 4, 187217 (1983) http://www.ch.embnet.org/MD_tutorial/.

28. Stewart, W. Introduction to Molecular Modeling. Stewart's Thesis, [Online]Available,

http://cat.middlebury.edu/ chem/chemistry/students/williamson/stewartdocs/thesis.html, November, 2004.

29. Mackerel 1995, phys. Chem. 102, 3586.

30. Nicholas Breen, Conformational Search: M.D. M.C. and S.D. [Online]Available, http://lungo.chem.columbia.edu/biophys_2002/search/backgroundII.html , November, 2004.

31. protein dynamics simulations from nanosecond to microseconds by Sebastian D, Peter E. Elsevier Science 1999.

32. A. Amadei,A.B.M Linssen, and H.J.C. Berendsen. Essential dynamics of proteins. Protein: Structure, Function and Genetics, 17(4):412-425, 1993.

33. Doniach, S. Eastman, P.. Protein dynamics simulation from nanoseconds to microseconds. Elsevier Science 9:157-163(1999).

34. Warmels, R.H.: , Principal Components Analysis "multivarient Analysis " [Online]Available, http://www.eso.org/projects/esomidas/doc/user/98NOV/volb/node212.html, November, 2004.

35. Norris. V , Multivariate Analysis and Reverse Engineering of Signal Transduction Pathways, Chapter 2 - Multivariate analysis, [Online]Available, http://www.iam.ubc.ca/ norris/research/amythesis.pdf, November, 2004.

36. MAM1

37. Kabasch, W. A solution for the best rotation to relate two sets of vectors. (1978) A34, 827-828.

38. Nicholas M. Glykos, presently at FORTH, IMBB, Heraklion, Crete, Greece. Please send comments, suggestions and bug reports to glykoscrystal2imbbforth.gr or glykos@imbb.forth.gr.

39. Student Tutorials, COSC453 2004 "chapter 3", [Online]Available, http://www.cs.otago.ac.nz/cosc453/student_tutorials/principal_components.pdf. November, 2004.

40. Hnenberger, P.H., Mark, A.E. van Gunsteren, W.F., Fluctuation and cross-correlation analysis of protein motions observed in nanosecond molecular dynamics simulations. J. Mol. Biol, 252, 492-503 (1995).

41. C.D. Schwieters, J.J. Kuszewski, N. Tjandra and G.M. Clore, Guide of The Xplor-NIH NMR Molecular Structure Determination Package, J. Magn. Res., 160, 66-74 (2003).

42. Humphrey, W., Dalke, A. and Schulten, K., ”VMD - Visual Molecular Dynamics”, J. Molec. Graphics, 1996, vol. 14, pp. 33-38. ¡http://www.ks.uiuc.edu/Research/vmd/¿

43. Walters, P. Stahl, M. Babel File Format Converter, [Online]Available, http://www.osc.edu/PET/CCM/skeleton/software/tested/ms-dos/babel/babel.html, November, 2004.

44. Butler, D, Basics of the Tcl Language, [Online]Available, http://www.csc.calpoly.edu/ dbutler/tutorials/winter96/tcl/tclbasics.html. November, 2004.

45. Gullingsrud. J, CatDCD - Concatenate DCD files. [Online]Available, http://www.ks.uiuc.edu/Development/MDTools/catdcd/. November, 2004.

46. Phillips, J. FlipDCD DCD file endianism converter. [Online]Available, http://www.ks.uiuc.edu/Development/MDTools/flipdcd/ November, 2004.

47. Faber, K. On Solving Generalized Eigenvalue Problems Using MATLAB Journal of Chemometrics, v 11, n 1, 1997, p 87, Compendex.

48. MacKerell, A. University of Maryland: CHARMM Empirical Force Fields Online]Available, http://www.pharmacy.umaryland.edu/faculty/amackere/param/toppar/toppar_c31b1.tar.g November, 2004.

49. The Board of Trustees of the University of Illinois, Building Gramicidin A. [Online]Available, http://www.ks.uiuc.edu/Research/namd/tutorial/NCSA2002/hands-on/main.html. Mon, Aug 19 2002.

50. Stonebank. M. UNIX Tutorial Two, [Online]Available, http://www.ee.surrey.ac.uk/Teaching/Unix/unix2.html. October 2002.

51. M. F. van Aalten, A. Amadei, A. B. M. Linssen, V. G. H. Eijsink, G. Vriend and H. J. C. Berendsen. The essential dynamics of Thermolysin: confirmation of the hinge-bending motion and comparison of simulations in vacuum and water. Proteins: Structure, Function, and Genetics. 22, 45-54 (1995).

52. Proteins. Structures and Molecular Properties" T.E. Creighton, W.H. Freeman Co., New York (1984) Chapter 6 pp 204-220. http://adelie.biochem.queensu.ca/ rlc/teaching/definitions.shtml, November, 2004.

53. W. van Gunsteren. Validation of molecular dynamics simulation. . Comput. Phys., 108:6109-6116, 1998.

54. Karplus M, Petsko GA. Molecular dynamics simulations in biology. Nature 347(6294):631-9 (1990).

55. Amadei, A. B. M. Linssen and H. J. C. Berendsen. Essential dynamics of proteins. Proteins: Structure, Function, and Genetics, 17, 412-425 (1993).

56. Balsera M.A, Principal Component Analysis and long time protein dynamics. J. Phys. Chem. 100:2567-2572 (1996).

57. Luo, J. Bruice, T. C.. Ten nanosecond molecular dynamics simulation of the motions of the horse liver alcohol dehydrogenasePhCH2O-complex. Proc. Natl. Acad. Sci (USA), 2002, 99, 16597-16600.

58. Kitao, A,Nobuhiro Go. Investigating protein dynamics in collective coordinate space. Elsevier Science 9:164169(1999).