Experimentally Biased Molecular Dynamics Simulations Using Neutron Reflection Data

by
Bradley W. Treece

Submitted in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy
at
Department of Physics
Carnegie Mellon University
Pittsburgh, Pennsylvania

Committee:
Mathias Lösche
Frank Heinrich
Arvind Ramanathan
Markus Deserno

April 28, 2020

Abstract

Structural modeling of proteins and characterization of their equilibrium properties is an essential part of understanding the complexities of organic matter. In addition, a wide array of biological functions occur at or are regulated by a lipid membrane. For this reason, characterization of proteins at bilayer interfaces is a crucial research pursuit. Neutron reflectometry (NR) is a powerful experimental technique for characterizing some of the interaction of proteins with lipid bilayers. Molecular dynamics (MD) simulations provide a computational framework to model details beyond what is accessible in experiment. Traditionally, these two methods are performed independently and their results are used in complement to one another.

In this work, a new method is developed and explored in order to integrate NR data into MD simulations. The algorithm takes the real–space model of the protein derived from NR scattering data and compares it to the corresponding data calculated from simulation, updating this information as the simulation progresses. This comparison is used to construct a potential that biases the protein's dynamics in order to garner better agreement between the MD and NR data.

First, a review is conducted of MD simulations, NR scattering, and the specific experimental systems under investigation. This review gives a general sense of how other experimental techniques and MD simulations are integrated and how NR data may be used in simulation. Next, an analysis of the effect biasing has on the dynamics of a system is explored. This motivates the use of a linear bias potential.

After supplying motivation for the method, details of the implementation are presented. The linear difference of the NR and MD data is used in the potential, and the MD data is calculated using two methods. In one method, the data is calculated only taking into account the protein conformation locally in time (memoryless bias). In the other, the running average of the data is tracked and used in the comparison to NR (historied bias). The effectiveness of these two methods is explored on two model systems.

The first system is a small helical peptide that has degrees of freedom equivalent to a rigid rod, under the energy scales explored. This system is not studied using neutron data, but rather a highly artificial potential is generated to represent only a confined subset of available conformations. Since the data is a reduction of the structure from three dimensions to one, the effects of the bias are not known before the simulations are performed.

However, the results of the memoryless bias simulations show that it is able to confine the peptide to a set of conformations like those used to generate the data. The historied bias initially behaves similarly, but as time progresses, the set of explored conformations becomes larger than in the memoryless bias. The agreement is better than in an unbiased simulation of the peptide.

The second system studied is of the PTEN tumor suppressor, which was previously characterized by NR and MD in a non-integrated way. The agreement of unbiased simulations is already significant, as shown by this previous work. The memoryless bias again succeeds in improving the agreement by making small modifications to the arrangement of domains within the protein. However, using too strong of a bias suppresses fluctuations and adds distorting stress to structured parts of the protein. In the historied bias simulations, the smaller applied bias leads to a different conformation which shows agreement that is almost as good as the stressed structure, without that stress. The larger applied bias initially agrees well but eventually leads to a third conformation that is in worse agreement than even the unbiased simulation. Both of the historied biases show less fluctuation in their conformations than is expected based on the helical peptide simulations. Additionally, both historied biases have portions of the tail associating with the membrane, while none of the others display this behavior.

After the results of these simulations, some modes of adding complexity to the algortithm are explored. In motivating these modifications, the time averaging of MD data in the historied bias is shown to weaken the effect of the bias in repelling the system from its current state. Taking this into consideration, the reduced conformational fluctuations in the PTEN historied bias simulations and the large disagreement at the larger bias suggest that the membrane–tail interaction may produce a meta–stable state. The larger memoryless bias could not have allowed the tail to find the membrane, and the smaller bias or unbiased simulation may have taken some time to discover this conformation. A biologically interesting result for PTEN was not expected, but an avenue for further work was uncovered, especially based on conformations seen in the smaller historied bias.

Acknowledgments

I would like to thank my advisor Dr. Mathias Lösche for his support and help as I took on the daunting task of a thesis project that was outside the standard research performed in his lab. I want to thank him for welcoming me into his lab, being patient, and keeping me grounded as I figured out how to make the whole thing come together. I would also like to thank Dr. Frank Heinrich for his continual support and endless supply of rebuttals that forced me to sharpen my thoughts on the project. He generously included me on another related project that was very intellectually stimulating. Thank you to Dr. Arvind Ramanathan for providing me the opportunity to travel to Oak Ridge National Laboratory to get this project off the ground and for his technical advice over the years. The summer at ORNL was one which I will never forget. Thank you to Dr. Markus Deserno for always providing sharp insights that never fail to deepen my thoughts or illustrate where I have not quite grasped the root of a problem.

A warm thank you to Dr. Stephanie Tristam-Nagle and Dr. John Nagle for providing me with a research group when I started at CMU and for the inclusion on so many great papers. A special thanks to Dr. Tristam-Nagle for her frequent and unwaivering interest/support in my development through the program. I will always be grateful and fondly remember the trip to Cornell's synchrotron.

Thank you to the entire biophysics group at CMU for stimulating conversations and great fellowship as we pursued our different but similar research aims. A special thanks to Dr. Amy Stetten for working with me after hours on a very interesting fluid mechanics problem. Thank you also to Dr. Stephen Garoff for including me on the project and its eventual publication. It is one of my favorite problems to have worked on. With an infinite interest in all things mathematical that I can relate to and for the unbounded kindess he always provides, I would like to thank Dr. O. Teoman Turgut. He has truly been a valued friend and mentor.

Thanks to all of my friends in the program, those who came before and after me. Notable among them are Dr. Amy Stetten, Dr. Zachary McDargh, Dr. Rebecca Eells, Dennis Michalak, Chris Kervick, and Samuel Foley. However, there are so many people with so many colorful stories; there are far too many to list here. I would, however, like to acknowledge my thanks for all who participated in "Manfred United", both competitively and recreationally. It was truly a blast to play soccer with all of you.

I would like to thank all my non-CMU friends behind the scenes who I have known for much longer. Your continued support through this endeavor is immeasurable. Additionally many staff in the department have been vital, not only for the day to day things but for their injection of sanity into my research–addled brain. Among them are Mary Jane Hutchison, Amanda Bodnar, Eric Day, Chuck Gitzen, Hilary Homer, and Heather Corcoran.

Finally I would like to thank my family and Emma Oxford. None of this would have been possible without them. They have always supported me with boundless enthusiam. Emma has been the encouragement I needed at times when I thought it would not be possible to continue.

Contents

Chapter 1

Introduction

1.1 Simulations

Many types of computer simulations try to model and predict the behavior of proteins; ranging from fully time-resolved simulations (molecular dynamics - MD) to random sampling schemes (Monte Carlo). These modelling techniques give information that is complementary to both experimental and theoretical techniques. Rarely are either able to provide a full picture of a biological process in its entirety, even when taken together. Computer simulations are often able to fill in the gaps that other techniques may leave behind. However, within subcategories of computer modelling there is a wide range of simulation tools and techniques at one's disposal. For example, MD simulations can represent a range of resolutions for a given molecule - from the atomistic to a representation of whole chemical groups as single beads (a generalized atom). Careful consideration should be made when choosing the correct tool for the question one would like to answer.

1.1.1 How Atomistic Simulations Are Performed

Atomistic MD simulations track the dynamics of every atom in the system. The interactions are governed by a classical force field that attempts to approximate the behavior of real systems, therefore requiring tuning of the force field to achieve observables that are in alignment with experimental data. Many different force fields exist (CHARMM, Amber, GROMOS, etc.), each parametrizing the interactions in a slightly different way.

The typical interactions accounted for in atomistic simulations include both nonbonded interactions between all atom pairs (electrostatic, Van der Waals) and bonded interactions between neighboring atoms (bonds, angles, dihedrals). These interactions dictate (classical) equations of motion and are numerically integrated to obtain the dynamics of the system. The typical nonbonded interactions are given in eqs (1.2) and (1.3), three of the typical bonded interactions are included in eqs (1.4-1.6).

$$U = U_{elec} + U_{vdw} + U_{bond} + U_{ang} + U_{dihe} + \dots \tag{1.1}$$

$$U_{elec} = \sum_{\text{nonbonded pairs}} \frac{q_i q_j}{4\pi\epsilon r_{ij}} \tag{1.2}$$

$$U_{vdw} = \sum_{\text{nonbonded pairs}} \varepsilon_{ij} \left[\left(\frac{R_{ij}^{\min}}{r_{ij}} \right)^{12} - 2 \left(\frac{R_{ij}^{\min}}{r_{ij}} \right)^{6} \right] \tag{1.3}$$

$$U_{bond} = \sum_{\text{bonds}} k_b (b - b_0)^2 \tag{1.4}$$

$$U_{ang} = \sum_{\text{angles}} k_\theta \left(\theta - \theta_0\right)^2 \tag{1.5}$$

$$U_{dihe} = \sum_{\text{dihedrals}} k_\phi \left(1 + \cos(n\phi - \delta)\right) \tag{1.6}$$

Other interactions are included to provide various corrections, depending on the force field of choice. Some of these are the CMAP, Urey-Bradley, and improper dihedral interactions. In addition to the interactions, it is often desired to have the dynamics evolve in an NPT ensemble (constant particle number, pressure, and temperature). To this end, the system must have control mechanisms for temperature (thermostat) and pressure (barostat). A few choices for thermostat include rescaling of the velocities as the temperature fluctuates[1], the Berendsen *weak-coupling* method[2], the extended ensemble Nosé-Hoover scheme[3,4], and the stochastic randomization method of Andersen[5]. Some proposed pressure control mechanisms are that of Berendsen[2], the *extended–ensemble* method of Parrinello–Rahman[6] which is a generalization of the method by Andersen[5], the MTTK implementation[7] (Martyna-Tuckerman-Tobias-Klein), and the *Langevin piston* method[8]. The

Langevin piston Nosé–Hoover method in NAMD combines the last two methods, called Nosé–Hoover since the MTTK pressure control is an extension of the work done by Nosé and Hoover to construct NPT ensembles[3,4].

Careful choice of a thermostat or barostat should be made depending on the nature of the observables under study. Velocity randomizing algorithms can alter dynamical observables like diffusion constants and shear viscosity, since such randomizations affect velocity correlations among particles.[9] The *extended–ensemble* has volume fluctuations which may contain ringing oscillations depending on the stiffness of the coupled system.[8] In addition, it is important for statistical averaging that the system be ergodic and certain schemes have been developed to achieve this end, such as the chaining of multiple thermostat variables in the *Nosé–Hoover chains* implementation.[10]

1.1.2 Sampling In Atomistic Simulations

The main advantage of atomistic simulations is also its biggest disadvantage: every atom is accounted for. Because the resolution is finer, the computational load is much larger than for a coarse grained simulation. The number of interactions that must be taken into account increases as complexity is added to the model. This consequently leads to long run times for the simulation that provide only a short glimpse of the dynamics and a large data set. MD simulations consistently fall short of what is needed for statistically valid equilibrium sampling.[11,12] Many biomolecular processes occur on the millisecond timescale or even much longer.[13] If statistical analysis of observables is desired, then many uncorrelated samples will be required. This increases the computational load even further, since MD simulations by their nature have correlation between adjacent configurations in their trajectories.[14]

To address the sampling issue, many different modes of simulation have been developed to find as many uncorrelated equilibrium configurations as possible. Replica exchange takes advantage of the temperature dependence of the Boltzmann factor in the probability to reach configurational transitions by having a number of replicas at different temperature that can be exchanged to allow both transitioning and sampling at appropriate temperatures.[15,16] Similar to replica exchange is simulated annealing, where high temperature promotes the trajectory to traverse a large distance through configurational space while throughout, many branching trajectories are constructed along the main trajectory. In the branches, the temperature is stepped down to those relevant for biological processes. This 'anneals' the offshoots into a

number of distinct configurations at the relevant temperature.[17–19]

Modifying the other part of the Boltzmann factor in the configurational probability, Hamiltonian exchange is a technique that alters the interaction potential in order to increase the rate of transition to different parts of configurational space. For instance, energy landscapes can be smoothed using models with 'softened' van der Waals interactions,[20] but altering the potential requires reweighting of the samples to correctly represent the probability distribution.[21,22] Many variations on temperature and Hamiltonian based smoothing exist. For example, the two can be combined by employing an exchange scheme with replicas that have either altered temperatures or smoothed Hamiltonians, producing a 'multidimensional' approach to replica exchange.[23]

Rather than modify the probability distribution to generate more distinct samples, the algorithm generating trajectories can be optimized. For example, integration of the equations of motion for all independent degrees of freedom need not be performed with the same time step if there is a large variance in the timescales associated with those degrees' motions. An integrator with different time steps for faster and slower degrees of freedom can speed up the integration and lead to faster sampling of independent configurations.[24,25] Similar to removing time resolution in slow modes, removing spatial resolution can also be useful by reducing the number of interactions to calculate. A number of methods for treating simulations with multiple resolutions have been constructed, as well as ways to reincorporate the full atomic resolution.[26–29]

Langevin dynamics simulations reduce computation loads by exchanging the dynamical trajectory with a Brownian-like motion of the protein atoms.[30–34] Markov Chain Monte Carlo simulations follow a similar strategy of reducing computation by avoiding the force calculation. Sequential configurations in the trajectory are based on a random proposed change that is subject to a probabilistic acceptance criterion. This criterion is defined using the energy difference between the configurations.[35–37] If the trial moves are small, it has been shown that the Monte Carlo dynamics approximate Langevin dynamics[35,38] and that quasi-physical moves can be more efficient than molecular dynamics[37,39], especially if those proposal moves are chosen from libraries of configurations for each amino acid.[40]

1.1.3 Using Experimental Data In Simulation

In addition to computational expense, another issue with simulations are the force fields. There are many different force fields to choose from and none of them perfectly capture the equilibrium properties or dynamics of a system,[41] primarily because internal interactions are classically approximated within the parameters of the force field. Due to the large number of atoms in a simulation and nonlinear dynamics inherent in the equations of motion, small differences between interactions in the simulation and those in the real world are difficult to predict or track. This makes tuning of force fields very difficult, if not impossible, to perfect. Because perfecting force fields is so daunting, other ways of correcting the simulation must be considered. One way to approach this is through the application of restraints based on experimental data. With every experimental technique aimed at structural refinement, how to model the data using a numerical approach is an often fruitful question to be asked.

For example, the molecular dynamics flexible fitting (MDFF) protocol takes cryo-EM density data and creates a coulombic potential energy (U_{EM}) which is added to the existing interaction energies in the MD engine, also adding secondary structure preserving harmonic potential (U_{SS}) to prevent overfitting.[42,43] The MDFF protocol has been extended to include x-ray crystallography data (xMDFF). However, the potential periodically requires structural information from the simulation to update the phase information of the density used in constructing U_{EM}.[44] The MDFF protocol was further modified with the release of iMDFF, a tool for performing user-guided fitting of structures into the experimental densities instead of steering molecular dynamics simulations with the data. In iMDFF, haptic feedback from a virtual reality capable controller provides guidance in the manipulation of protein structures based on their compatability with X-ray or cryo-EM data.[45,46]

Nuclear magnetic resonance (NMR) provides expected values for various interatomic distances across a protein and dipole couplings in NMR give the relative orientations of atom groups in the protein, both of which can be used to restrain the configurations accessed in a simulation.[47–49] Small or Wide Angle X-ray Scattering (SAXS, WAXS, SWAXS) experiments provide scattering profiles that can be calculated throughout a simulation run and provide biasing via an energy functional.[50,51] An aggregation of various modes of experimental biasing through the restraining of 'collective variables' has been implemented for the distances derived from NMR, other positional or

orientational deviations from a reference that may have experimental data available, and the self coordination number of solute molecules.[52]

In addition to simulation-based procedures, a number of Bayesian or iterative approaches have been developed to refine large generated sets of configurations using experimental data. A few approaches in this camp for NMR data are KNOWNOE[53] and Protinfo[54] for data from the nuclear Overhauser effect and CHESHIRE[55], CS-Rosetta[56], and CS23D[57] for NMR chemical shift data. For X-ray crystallography and cryo-EM, similar techniques include DireX[58], Flex-EM[59], Rosetta[60,61], and FRODA[62]. Combining these approaches of structure refinement through the minimization of a scoring function (or maximizing a likelihood) for cryo-EM, X-ray, and NMR, as well as mass spectometry[63], mutagenic studies[64], and other experimental techniques to collectively score structure is a technique which can be described using the umbrella term 'integrative modeling'.[65–72] This field is vast and not considered here in detail, as the research presented later will be in the category of experimentally driven dynamical simulations and not integrative modeling.

1.2 Neutron Reflectivity

Neutron Reflectivity (NR) is an elastic scattering technique that is capable of probing a stratified sample. The details of NR are readily available[73,74] and so only the basic theory will be discussed in the following. Neutrons have particle-wave duality, thus a wave scattering theory can be applied to them. Their wavelength will depend on the reciprocal of their kinetic energy. For NR, neutrons are reflected specularly from the layers of the sample, leading to a momentum transfer normal to the interface of the sample (see figure 1.1). Specular reflection obeys Snell's Law, implying that the angle of incidence and reflection are equal. For this reason, the normal momentum transfer is twice the normal component of the incident momentum:

$$q_z = \frac{4\pi}{\lambda}\sin(\theta) \tag{1.7}$$

where λ is the wavelength of the neutrons and θ is the incident (and reflected) angle. In addition to reflected neutrons, there are neutrons which are transmitted into the sample. Subsequent strata of the sample will have reflection and transmision at their interfaces as well. It is the series of reflections (see

figure 1.1) that lead to the interference patterns observed in NR; analysis of this pattern yield information about the layers of the sample.

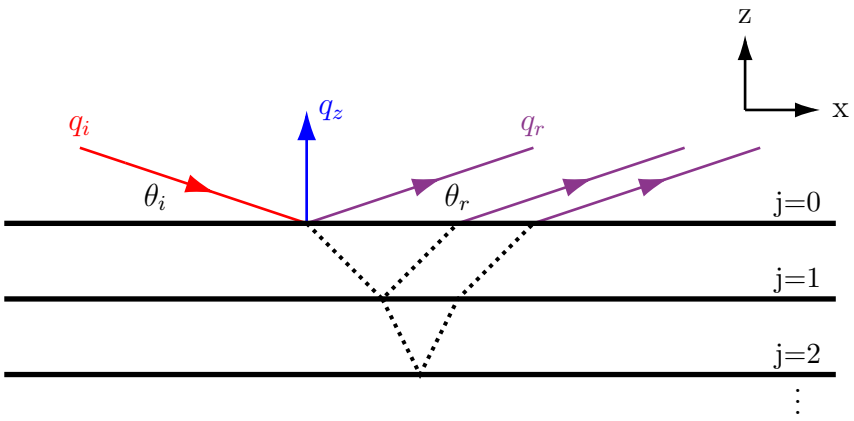

Figure 1.1: Geometry for the elastic scattering of neutrons from a stratified sample. At the first interface, a wave of incident neutrons (red) is reflected (purple) and transmitted (dotted black) into subsequent layers to be transmitted and reflected at each interface encountered. Reflected waves that are transmitted out of top of the sample all have a change in momentum (blue) as given in eq (1.7).

The transmission and reflection will depend on the refractive indices of the various layers. The neutron refractive index for a given medium is defined to be

$$n = 1 - \frac{\lambda^2 N_d\, b}{2\pi} + \frac{i\lambda N_d\, \sigma_a}{4\pi} \tag{1.8}$$

N_d is the atomic number density, b is the coherent scattering length, and σ_a is the absorption cross section. For most materials, $\sigma_a \approx 0$ and the second term is irrelevant. The neutron scattering length density (nSLD) is defined as

$$\rho = N_d\, b \tag{1.9}$$

Substituting eq (1.9) into the definition for the refractive index under the assumption that the cross section vanishes gives, for n,

$$n = 1 - \frac{\lambda^2}{2\pi}\rho \tag{1.10}$$

As mentioned before, Snell's Law implied an equality between the reflected and incident angle, but it also relates the incident angle in one medium to the transmitted angle in the next medium.

$$n_1 \cos(\theta_1) = n_2 \cos(\theta_2) \tag{1.11}$$

1.2.1 Reflection from a single interface

Assume that neutrons are incident from air ($n_1 = 1$) and strike a sample with $n_2 = n$. For most materials, the scattering length is positive and the index of refraction will be less than that of air. Protium is a special case, having negative scattering length. For this reason, common water (undeuterated) has an index greater than one and behaves qualitatively different than most materials, making it useful for scattering. But, for the majority of interfaces with air, Snell's Law cannot be satisfied with too small of an incident angle. This is a well known phenomenon known as total internal reflection. In this case, all of the neutrons are reflected and none are transmitted. The critical angle, above which Snell's Law holds, depends only on the refractive index of the sample, given the conditions stated and the largest transmitted angle allowed ($\theta_2 = \pi/2$).

$$\cos(\theta_c) = n \tag{1.12}$$

For small enough θ_c, a Taylor expansion gives

$$\cos(\theta_c) \approx 1 - \frac{\theta_c^2}{2} \tag{1.13}$$

Combining eqs (1.10) and (1.13) relates the critical angle to the nSLD and wavelength:

$$\theta_c = \lambda \sqrt{\frac{\rho}{\pi}} \tag{1.14}$$

The critical momentum transfer can be found by Taylor expanding eq (1.7) and substituting in eq (1.14).

$$q_c \approx \frac{4\pi}{\lambda}\theta_c = 4\sqrt{\pi\rho} \tag{1.15}$$

For wave vectors smaller than the critical wave vector, all of the neutrons are reflected. Above the critical angle, the reflectance and transmittance

are the same as for electromagnetic waves, defined as the square modulus of the ratio between the reflected (or transmitted) wave amplitude and the incident wave amplitude. Fresnel's equations give that the reflectance is, incorporating the indices of air and the sample,

$$R = \left| \frac{\sin\theta_i - n\sin\theta_t}{\sin\theta_i + n\sin\theta_t} \right|^2 \tag{1.16}$$

Referring to eq (1.11),

$$n\sin(\theta_t) = \sqrt{n^2 - n^2\cos^2(\theta_t)} = \sqrt{n^2 - \cos^2(\theta_i)} \tag{1.17}$$

The index of refraction is related to the critical angle through eq (1.12).

$$\begin{aligned} \sqrt{n^2 - \cos^2(\theta_i)} &= \sqrt{\cos^2(\theta_c) - \cos^2(\theta_i)} \\ &= \sqrt{\left(1 - \sin^2(\theta_c)\right) - \left(1 - \sin^2(\theta_i)\right)} \\ &= \sqrt{\sin^2(\theta_i) - \sin^2(\theta_c)} \end{aligned} \tag{1.18}$$

The reflectance can then be expressed entirely in terms of the incident momentum transfer vector and the critical vector using eq (1.7) and eq (1.18).

$$\begin{aligned} R &= \left| \frac{\frac{\lambda q_z}{4\pi} - \sqrt{\left(\frac{\lambda q_z}{4\pi}\right)^2 - \left(\frac{\lambda q_c}{4\pi}\right)^2}}{\frac{\lambda q_z}{4\pi} + \sqrt{\left(\frac{\lambda q_z}{4\pi}\right)^2 - \left(\frac{\lambda q_c}{4\pi}\right)^2}} \right|^2 \\ &= \left| \frac{1 - \sqrt{1 - \left(\frac{q_c}{q_z}\right)^2}}{1 + \sqrt{1 - \left(\frac{q_c}{q_z}\right)^2}} \right|^2 \end{aligned} \tag{1.19}$$

The intensity of the reflected neutrons is identical to the incident beam below the critical angle. Above the critical angle, the intensity decays as more neutrons are transmitted through the interface (see Fig 1.2). As mentioned, water does not contain a critical angle feature and begins to decay at $q_z = 0$.

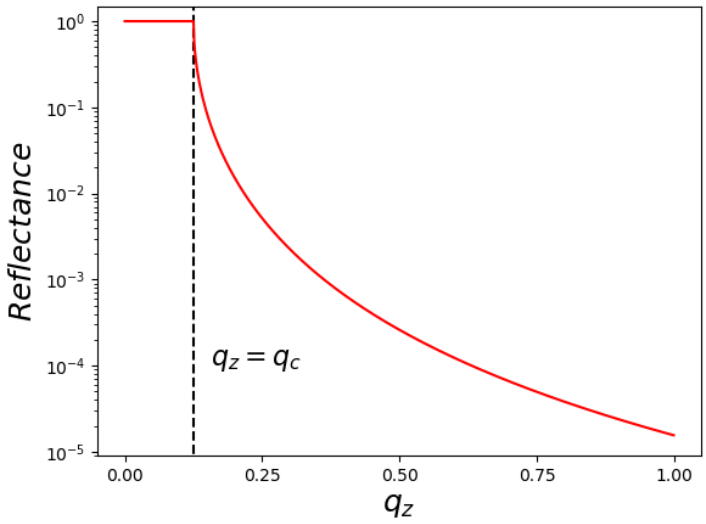

Figure 1.2: Fresnel reflectance versus momentum transfer for neutrons incident on a single interface, given by eq (1.19). A logarithmic scale is used for the reflectance axis. Below the critical momentum (q_c), all neutrons are reflected. Above the critical momentum, more neutrons are transmitted through the interface as the incidence angle increases.

1.2.2 Reflection from multiple interfaces

In the case of a stratified medium (see figure 1.1), multiple reflections and transmissions of the neutron beam occur at each interface. In each layer there is an ingoing and outgoing wave, except for the last medium (barring total reflection) where the transmitted wave continues to infinity. Writing down the incoming and outgoing neutrons as plane waves and orienting the axes such that the plane of incidence is the x-z plane (figure 1.1),

$$\psi_{in} = A_j^+ e^{i(\omega t + k_{z,j} z + k_{x,j} x)} \tag{1.20}$$

$$\psi_{out} = A_j^- e^{i(\omega t - k_{z,j} z + k_{x,j} x)} \tag{1.21}$$

Here, k is used to represent the wave vector with x and z components ($k = \sqrt{k_x^2 + k_z^2} = 2\pi/\lambda$) which varies throughout layers with the medium and the subscript j denotes which layer of the sample is being considered (figure 1.1).

The wave vector changes because the potential for the neutrons varies in z, as the composition of the medium changes. For this reason, only the component of the momentum normal to the layers changes at an interface, the parallel components are continuous across the interface[73]. It is, therefore, useful to break up the wave function into a z dependent part, the total wave function is given by the sum of the incoming and outgoing waves.

$$\psi = \left(A_j^+ e^{ik_{z,j}z} + A_j^- e^{-ik_{z,j}z}\right) e^{i(\omega t + k_{x,j}x)} = \left(\psi_\perp(k_{z,j}, z) + \psi_\perp(-k_{z,j}, z)\right)\psi_\parallel(x,t) \tag{1.22}$$

The wave function and its derivative must be continuous across an interface. The $\psi_\parallel$ portion trivially satisfies this and drops out of the boundary condition at each interface. The remaining portion of the boundary condition at the interface between layers j and $j+1$ (at depth $Z_{j,j+1}$) are given by:

$$\begin{aligned}\psi_\perp(k_{z,j},&Z_{j,j+1}) + \psi_\perp(-k_{z,j}, Z_{j,j+1}) \\ &= \psi_\perp(k_{z,j+1}, Z_{j,j+1}) + \psi_\perp(-k_{z,j+1}, Z_{j,j+1})\end{aligned} \tag{1.23}$$

$$\begin{aligned}k_{z,j}&\Big[\psi_\perp(k_{z,j},Z_{j,j+1}) - \psi_\perp(-k_{z,j}, Z_{j,j+1})\Big] \\ &= k_{z,j+1}\Big[\psi_\perp(k_{z,j+1}, Z_{j,j+1}) - \psi_\perp(-k_{z,j+1}, Z_{j,j+1})\Big]\end{aligned} \tag{1.24}$$

The first equation matches the wavefunction, the second matches the derivative. These equations are equivalent to those for an electromagnetic field that is s-polarized (perpendicular to the plane of incidence). The conditions can be compiled into a single matrix equation.

$$\begin{bmatrix} \psi_\perp(k_{z,j}, Z_{j,j+1}) \\ \psi_\perp(-k_{z,j}, Z_{j,j+1}) \end{bmatrix} = \begin{bmatrix} p_{j,j+1} & m_{j,j+1} \\ m_{j,j+1} & p_{j,j+1} \end{bmatrix} \begin{bmatrix} \psi_\perp(k_{z,j+1}, Z_{j,j+1}) \\ \psi_\perp(-k_{z,j+1}, Z_{j,j+1}) \end{bmatrix} \tag{1.25}$$

$$\begin{aligned} p_{j,j+1} &= \frac{k_{z,j} + k_{z,j+1}}{2k_{z,j}} \\ m_{j,j+1} &= \frac{k_{z,j} - k_{z,j+1}}{2k_{z,j}} \end{aligned} \tag{1.26}$$

The matrix above changes the magnitudes of the wavefunctions as the neutrons transition between media and is called the refraction matrix $\mathcal{R}_{j,j+1}$. Additionally, the wave function changes magnitude with position in the layers. It is therefore useful to construct a translation matrix $\mathcal{T}_j$ that accounts for the change in depth between each layer:

$$\begin{bmatrix} \psi_\perp(k_{z,j}, Z_{j-1,j}) \\ \psi_\perp(-k_{z,j}, Z_{j-1,j}) \end{bmatrix} = \begin{bmatrix} e^{-ik_{z,j}d_j} & 0 \\ 0 & e^{ik_{z,j}d_j} \end{bmatrix} \begin{bmatrix} \psi_\perp(k_{z,j}, Z_{j,j+1}) \\ \psi_\perp(-k_{z,j}, Z_{j,j+1}) \end{bmatrix} \quad (1.27)$$

The thickness of a given layer is denoted d_j. Starting at the last layer, the wavefunction vector is sequentially multiplied by the refraction matrix and the translation matrix for a layer until it can be related to the wavefunction vector at the first interface:

$$\begin{bmatrix} \psi_\perp(k_{z,0}, Z_{0,1}) \\ \psi_\perp(-k_{z,0}, Z_{0,1}) \end{bmatrix} = \mathcal{R}_{0,1}\mathcal{T}_1\mathcal{R}_{1,2}\ldots\mathcal{R}_{n-1,n} \begin{bmatrix} \psi_\perp(k_{z,n}, Z_{n-1,n}) \\ \psi_\perp(-k_{z,n}, Z_{n-1,n}) \end{bmatrix} \quad (1.28)$$

The product of these refraction and translation matrices yields a single matrix that describes the entire scattering event. This matrix is called the transfer matrix $\mathcal{M}$. The reflectance, written in terms of the transfer matrix,

$$R = \left|\frac{\psi_\perp(-k_{z,0}, Z_{0,1})}{\psi_\perp(k_{z,0}, Z_{0,1})}\right|^2 = \left|\frac{\mathcal{M}_{21}\psi_\perp(k_{z,n}, Z_{n-1,n}) + \mathcal{M}_{22}\psi_\perp(-k_{z,n}, Z_{n-1,n})}{\mathcal{M}_{11}\psi_\perp(k_{z,n}, Z_{n-1,n}) + \mathcal{M}_{12}\psi_\perp(-k_{z,n}, Z_{n-1,n})}\right|^2 \quad (1.29)$$

When no total internal reflection occurs, the wave makes it to the last, semi-infinite medium. In experiments, the substrate will have minimal penetration by the neutrons and is therefore treated as the semi-infinte medium. Equivalent to setting the last interface infinitely far away, $Z_{n-1,n} = \infty$, is setting its reflected amplitude to zero.

$$\psi_\perp(-k_{z,n}, Z_{n-1,n}) = 0 \quad (1.30)$$

This simplifies the expression for reflectance:

$$R = \left|\frac{\mathcal{M}_{21}}{\mathcal{M}_{11}}\right|^2 \quad (1.31)$$

In the single interface case discussed in the last section, the transfer matrix is equal to the refraction matrix of the interface. The medium was

assumed to be semi-infinite, meaning eq (1.31) holds. Calculating the reflectance using the transfer matrix method,

$$R = \left|\frac{m_{0,1}}{p_{0,1}}\right|^2 = \left|\frac{k_{z,0} - k_{z,1}}{k_{z,0} + k_{z,1}}\right|^2 \tag{1.32}$$

The Schrödinger equation dictates that the momentum in air (free space) is related to that in a medium by the following equation:

$$k^2 = k_{air}^2 - 4\pi\rho \tag{1.33}$$

The parallel components of momentum are continuous through the interface. This means that the normal component of the momentum changes

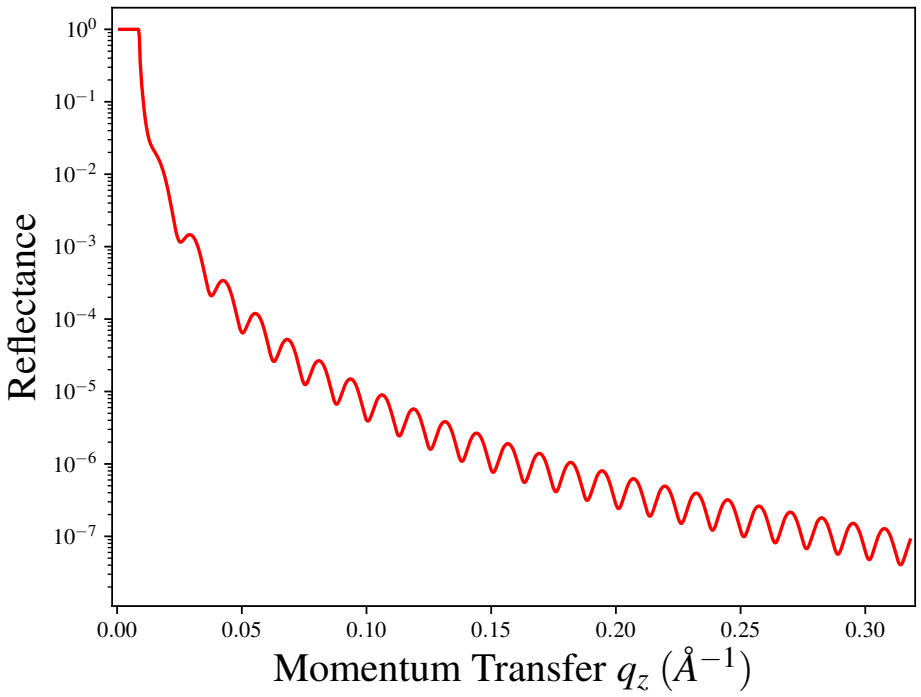

Figure 1.3: Modeled neutron reflectance curve for 500 $\AA$ of water ($\rho = -0.56 \cdot 10^{-6}$ $\AA^{-2}$) on a semi-infinite slab of silicon dioxide ($\rho = 1.58 \cdot 10^{-6}$ $\AA^{-2}$). The neutron wavelength used is monochromatic with $\lambda = 1$ $\AA$.

according to eq (1.33). This equation, eq (1.15), and that the normal component of the incident momentum is half the momentum transfer q_z, gives the following expression for the normal component of the wave vector:

$$k_{z,1} = \sqrt{k_{z,0}^2 - 4\pi\rho} = k_{z,0}\sqrt{1 - \left(\frac{q_c}{q_z}\right)^2} \tag{1.34}$$

Putting this relation into eq (1.32) gives an expression that simplifies to the expression obtained in eq (1.19). The reflectance for a 500 $\AA$ thick layer of water on Silicon as calculated from the matrix method is shown in figure (1.3).

1.3 Experimental Samples And Modeling

Sparsely tethered bilayer lipid membranes (stBLMs)[75] are biomimetic model membranes that have a stratified, planar geometry conducive to NR experiments (see figure 1.4). The structure of these membranes and their general neutron scattering profiles are well known.[76] This permits the study of how membrane-associated proteins interact with these bilayers.[77–84] The composition of these samples and how they are modeled in NR analysis will be discussed in the following sections.

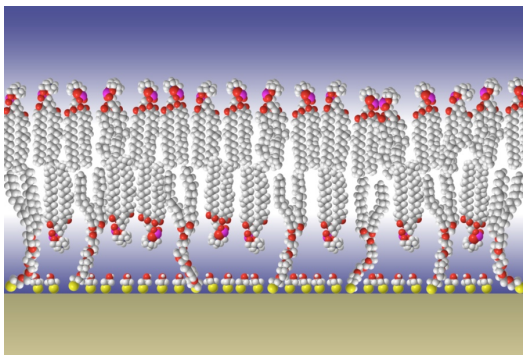

Figure 1.4: Sparsely tethered bilayer lipid membrane. A lipid bilayer is attached to a gold-coated substrate through a synthetic lipid tether. Additionally, the density of the tether is controlled through the use of a spacer molecule in order to "sparsely–tether" the membrane.

1.3.1 stBLMs

To construct an stBLM, a layer of chromium and then gold are sputter deposited onto a glass slide. The gold-coated substrate is covered in a mixture of lipid tethering molecules and spacer molecules. Each of these molecules contains a thiol group, which forms a bond with the gold substrate. The spacers are β-mercaptoethanol (βME), a complex of the thiol and ethanol, while the tethers contain a lipid-like structure. They are composed of a polyethylene chain of 6-9 units, followed by two hydrocarbon chains which can be saturated or unsaturated. These hydrocarbon chains integrate with the assembled bilayer on the inner leaflet (gold-side). The polyethylene chain gives a hydration layer, approximately 20 $\AA$ thick, between the bilayer and the gold. The spacers and tethers are typically in a ratio of 70:30, leading to a membrane which is sparsely tethered. This setup produces a flat bilayer with lipidic diffusion rates comparable to that in unilamellar vesicles.[85]

1.3.2 Component Based Model Analysis Of Neutrons

If an NR measurement of an stBLM is taken with or without protein, a scattering profile is produced which is much more complicated than the one shown previously in figure 1.3. The nSLD of the sample changes between the various layers of the substrate and between the various components of the biological sample. The bilayers even scatter differently whether in the headgroups or tail regions. This leads to an nSLD which varies a lot as a function of z (sample normal direction). The complexity of the nSLD profile leads to a scattering which is commensurately complex.

Given a scattering profile for an observed bilayer sample, backing out the underlying nSLD profile and subsequently the real space structure requires modeling and fitting of the scattering data. How the nSLD is modeled will be described later in this section. Real space modeling of sub-molecular components is accomplished by considering how cross-sectional area of each component varies as a function of z, the direction normal to the interface. The integral of this area profile over the entire span of the interface,

$$V_c = \int dz\ A_c(z) \tag{1.35}$$

gives the volume of the component, V_c. Figure 1.5 shows A_c for components of a DOPC lipid bilayer. The bilayer is subdivided into headgroups, tails,

and methyl groups.

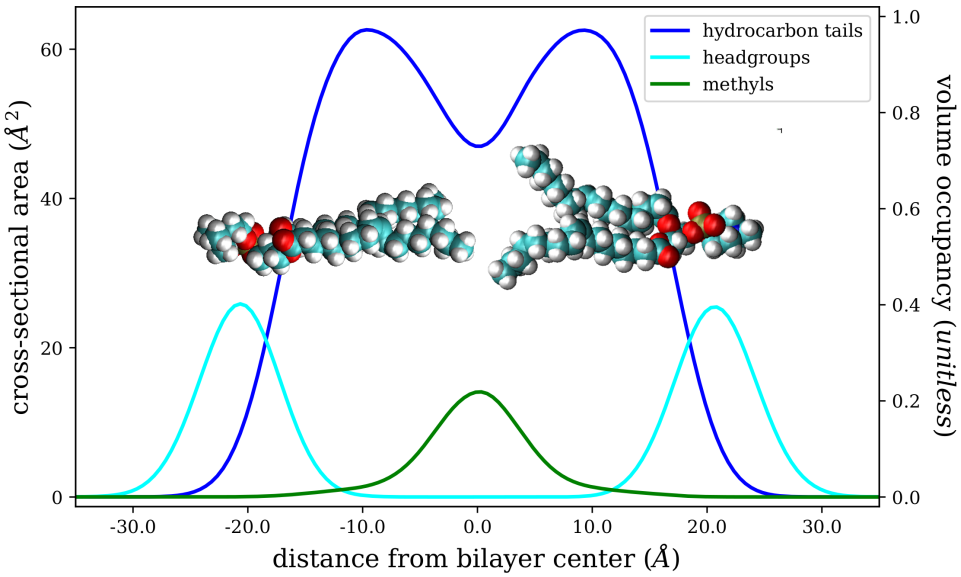

Figure 1.5: Cross-sectional area, $A_c(z)$, for components of a lipid bilayer as calculated from molecular dynamics simulation with a DOPC containing bilayer. The two molecules (overlaid) are subdivided into component profiles of headgroups (cyan), tails (dark blue), and methyl groups (green). Scale on the right shows equivalence to CVO profile after scaling the cross-section by the area per lipid (64.4 $Å^2$). Regions with summed components having their total CVO < 1 also contain other components, such as water.

A natural extension of the cross-sectional area profile is a profile detailing the amount by which a component fills space along the interface. For this, a component volume occupancy (CVO) profile is defined. The CVO sets a normalized, unitless scale (0-1) on which to compare the amount of space shared by different components, zero if there none of a particular component at some point and one if it is the only component at that point. As the axes indicate in figure 1.5, A_c and the CVO profile are proportional to one another and their dimensions differ by a factor of area. Given our interest in supported bilayer system, the area per lipid is a good choice. In the lipid

tail region, without protein inserted, the entire volume is occupied by the lipid. Also, the CVO for the protein will measure the bilayer coverage of the protein.

With the CVO profiles of real space components, a total nSLD profile can be constructed based on the scattering length for each component. Functionally, the nSLD is parametrized using a stratified slab model for the substrate[86] and using a system called the continuous distribution model for the components of the stBLM.[87] The parameters relevant to the slab model (step function) are the nSLD and thickness of each layer. The continuous distribution model, in contrast to the slab model, does not assume sharp cutoffs between the nSLDs of the various layers. There is a continuous transition between the nSLDs, so error functions are used instead. The error functions have a shape similar to the slabs but have an additional parameter, roughness, that creates a continuous transitional behavior between layers.

Figure 1.6 shows the neutron scattering, modeled nSLD, and modelled CVO for the PTEN experiments used later in this document. The parameters of the CVO were fit to produce the best agreement between the scattering observed and the scattering due to the nSLD resulting from the CVO fit. The protein CVO is determined by first performing a control fitting on a scattering experiment without protein present, to determine bilayer and substrate parameters, and subsequently fitting the scattering experiment with protein introduced to the same sample, using the bilayer and substrate parameters determined in the control. It is assumed that the protein does not significantly reorganize the rest of the sample.

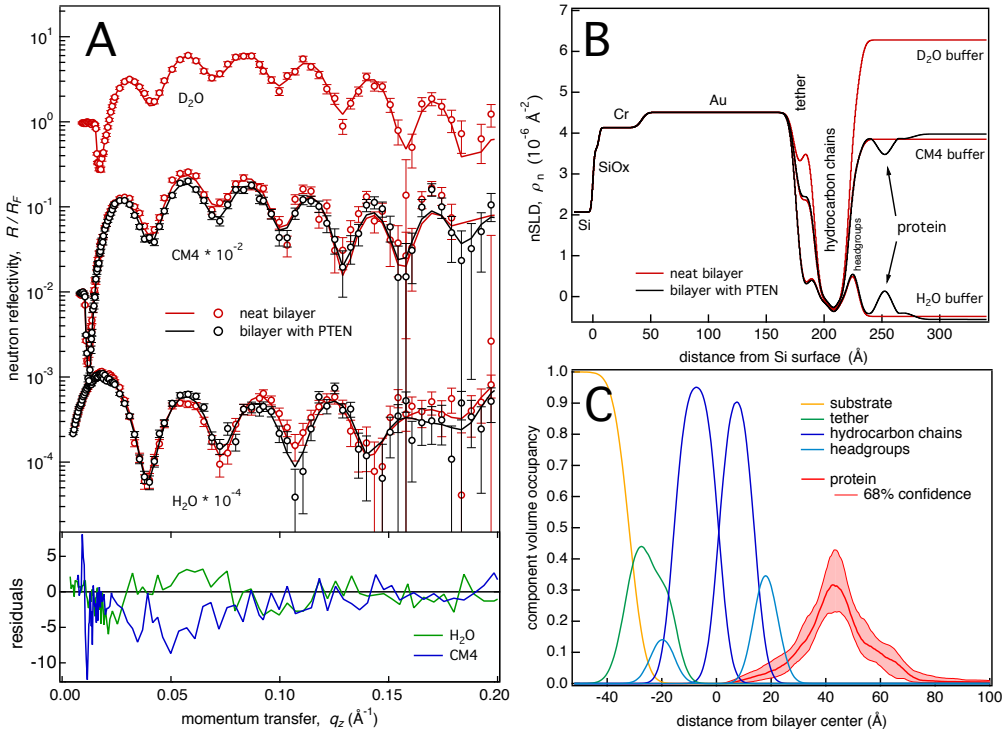

Figure 1.6: Example neutron reflection data of a bilayer on a gold–coated Si substrate before and after incubation with the PTEN tumor suppressor and their interpretation as a one-dimensional distribution of system components along the interfacial normal direction, z. (A) Neutron reflectivities of an stBLM (DOPC/ DOPS/chol = 67:30:3) in contact with buffers of different isotopic compositions (H_2O, D_2O and a mixture of the two with an SLD, $\rho_n \approx 4 \times 10^{-6} \mathring{A}^{-2}$, designated at 'CM4'). Black: As-prepared bilayer. Red: The same bilayer after incubation with PTEN.[88] While the differences in NR with and without protein are small, they are significant, as shown by the residuals plots on the bottom. The modeled reflectivities (solid lines) derive from simultaneous fits to 5 data sets measured under D_2O and H_2O based buffers. (B) Simultaneously optimized SLD profiles derived from a parameterized structural model expressed as a unique component volume occupancy (CVO) profile, shown in (C). The solid lines that fit the data in (A) originate from the SLD profiles shown in (B). The CVO profile, panel C, resolves the complex surface structure that contains the metal films on the Si substrate (at $z < 0$), the tether chemistry and lipid bilayer components of the stBLM (at $0 < z < 40$ $\mathring{A}$), and a protein layer (at $z > 40$ $\mathring{A}$) in which the PTEN phosphatase skims the membrane surface peripherally.

Chapter 2

Steered Simulations

As laid out in the introduction, difficulty in accurately modeling an enormous catalogue of experimental observables through simulations has led to the creation of many hybrid methods that seek to incorporate the physical data into the modeling directly. The focus of this chapter will follow one specific branch of these methods: biasing dynamical simulations through the use of energetic potentials that evaluate the correspondence observed between the experimentally measured values and those observed throughout simulation.

First, a theoretical framework is laid out for what it means to bias a simulation such that agreement with experiment can be reached, a clever method for doing so efficiently is examined, and then considered is the concept of minimally affecting the configuration space explored by the simulation. Later, a method for biasing using CVO profiles from neutron reflection is proposed. Finally, the results of simulations using this method on a toy model and an actual NR experiment is discussed.

2.1 Biasing MD Dynamics And Probability

The behavior of objects modeled in molecular dynamics simulations are dictated by the interaction potentials. The conformational states explored by the system when simulated under constant temperature are weighted by the Boltzmann factor. The probability of visiting any state is given by:

$$P(\vec{x}) = \frac{e^{-\beta U_0(\vec{x})}}{\int d\vec{x}\, e^{-\beta U_0(\vec{x})}} \tag{2.1}$$

Where U_0 is the interaction potential of the simulation, $\beta = k_B T$, and $\vec{x}$ is a generalization of any coordinates that may be of interest (not just position). Often, the classical force fields of a given molecular dynamics simulation fail to capture the exact statistical behavior of observables for real systems as force fields are merely an approximation of many interactions, with a large number of degrees of freedom to optimize. Often, many observables that are measurable in experiment are not fully in accordance with their values calculated from simulation, at least over time scales achievable given the computational resources available. If the observable of interest is $q(\vec{x})$ with experimentally determined value Q, accordance between experiment and simulation would be achieved under the condition:

$$\langle q \rangle = \int d\vec{x}\ q(\vec{x}) P(\vec{x}) = Q \tag{2.2}$$

where $P(\vec{x})$ is the probability distribution of states observed in simulation. The experiment and simulation are not in agreement when $\langle q \rangle \neq Q$. One way to deal with such a discrepancy is to add a restraining bias to the existing interaction potential that will drive the observable toward agreement with experiment. One easily implemented form of such a bias is to quadratically penalize any discrepancy between the simulation and the experimental value.

$$U_{bias}(\vec{x}) = U_0(\vec{x}) + \frac{k}{2}\left(q(\vec{x}) - Q\right)^2 \tag{2.3}$$

k sets the strength of the bias, also determining the amount to which the dynamics will vary from their native form. As will be demonstrated in the following example, this naive approach can indeed bring the observable into accordance. However, the other properties of the system may also be affected by such a bias.

2.1.1 Example: Restraint On A Particle In A Harmonic Potential

Take a particle under the influence of a harmonic potential,

$$U = \frac{p^2}{2m} + \frac{m\omega^2 x^2}{2} \tag{2.4}$$

Because we will not discuss anything that involves the momentum of the particle in the following and eq (2.1) is factorizable in terms of the momentum

contribution, the probability can be marginalized over p to be only a function of position.

$$P(x) = \sqrt{\frac{\beta m\omega^2}{2\pi}}\text{EXP}\left[-\frac{\beta m\omega^2}{2}x^2\right] \tag{2.5}$$

The normalization obtained from the integral over x is given in the prefactor. For this system, two observables are of interest in our discussion - the first and the centered second moments of position.

$$\langle x\rangle = \mu = 0 \tag{2.6}$$

$$\langle (x-\mu)^2\rangle = \sigma^2 = \frac{1}{\beta m\omega^2} \tag{2.7}$$

For the discussion, let us take the first moment, $\langle x\rangle$, as the experimental observable under consideration. If it is experimentally measured to be some nonzero value a, then a simulation with dynamics governed by eq (2.4) will disagree with experiment. Applying a restraining bias as described in eq (2.3) of the previous section the following potential energy and subsequent probability distribution are obtained.

$$U_{bias} = \frac{m\omega^2x^2}{2} + \frac{k}{2}(x-a)^2 = \frac{1}{2}\Big(\left(m\omega^2+k\right)x^2 - 2kax + ka^2\Big) \tag{2.8}$$

$$\begin{aligned} P_{bias}(x) &= \frac{\text{EXP}\left[-\frac{\beta}{2}\Big(\left(m\omega^2+k\right)x^2 - 2kax + ka^2\Big)\right]}{\int dx\ \text{EXP}\left[-\frac{\beta}{2}\Big(\left(k+m\omega^2\right)x^2 - 2kax + ka^2\Big)\right]} \\ &= \sqrt{\frac{\beta(k+m\omega^2)}{2\pi}}\text{EXP}\left[-\frac{\beta(k+m\omega^2)}{2}\left(x - \frac{ka}{k+m\omega^2}\right)^2\right] \end{aligned} \tag{2.9}$$

Eq (2.9) has been algebraically simplified to express the probability in a form for which the moments are easily identified.

$$\langle x\rangle_{bias} = \frac{ka}{k+m\omega^2} \tag{2.10}$$

$$\langle (x-\mu)^2\rangle_{bias} = \frac{1}{\beta(k+m\omega^2)} \tag{2.11}$$

With increasing k, the average position of the particle is shifted toward a, reaching an equality with it in the limit of large k (for which the probability distribution becomes a δ-function). This is precisely the desired behavior for that property: as the bias increases, the agreement between simulation and experiment increases. This agreement comes at a cost. The second moment is modified such that it grows ever smaller as the bias increases, vanishing in the large k limit. This is not suprising, since the bias is restraining the position ever more strongly as k grows. The hypothetical experiment did not prescribe any change to the second moment. Ideally, the restraint would modify the average position with minimal effect on other properties of the system.

2.1.2 Restrained Ensemble On The Harmonic Potential

As pointed out by Roux and Weare,[89] another method to bias a simulation is to apply the restraining potential to an ensemble of simultaneous simulations; using the average observable across those simulations instead of biasing any single instance independently. If there are N instances of the particle, each labeled by their own subscript i, the potential imposed on the particles is slightly modified.

$$U(\vec{x}) = \sum_i \frac{m\omega^2 x_i^2}{2} + \frac{k}{2}\left(\frac{1}{N}\sum_i x_i - a\right)^2 \tag{2.12}$$

Each instance of the particle is interacting with the harmonic well independently and interacts with the other instances through the bias term. The joint probability distribution for these particles, according to eq (2.1), is:

$$\begin{aligned} P(\vec{x}) &= \frac{e^{-\beta U}}{\int d\vec{x}\, e^{-\beta U}} \\ &= \mathrm{EXP}\left[-\frac{1}{2}\left(\beta m\omega^2 \sum_i x_i^2 + \beta k\left(\sum_i \frac{x_i}{N} - a\right)^2 + 2\ln(Z)\right)\right], \end{aligned} \tag{2.13}$$

where the partition function, Z, is absorbed into the exponential function of the numerator. Since the argument of the exponential is quadratic, the

probability is a multivariate normal distribution (MVN). An alternative expression for the MVN with mean vector $\vec{\mu}$ and covariance matrix $\boldsymbol{C} = \boldsymbol{S}^{-1}$ can be written:

$$\begin{aligned} P(\vec{x}) &= \frac{\mathrm{EXP}\left[-\frac{1}{2}\left(\vec{x}-\vec{\mu}\right)^T \boldsymbol{S}\left(\vec{x}-\vec{\mu}\right)\right]}{\sqrt{(2\pi)^N \left|\boldsymbol{C}\right|}} \\ &= \mathrm{EXP}\left[-\frac{1}{2}\left(\left(\vec{x}-\vec{\mu}\right)^T \boldsymbol{S}\left(\vec{x}-\vec{\mu}\right) + 2\ln(A)\right)\right] \end{aligned} \tag{2.14}$$

Again, the normalization factor A has been absorbed into the exponential function. It is distinct from the partition function since the Boltzmann factor in eq (2.13) is not the result of a perfect square, i.e. an extra constant term is added to the quadratic potential. Since both distributions are normalized, their quadratic forms can be compared term by term once expressed in a similar manner.

$$\begin{aligned} &\beta m\omega^2 \sum_i x_i^2 + \beta k\left(\sum_i \frac{x_i}{N} - a\right)^2 + 2\ln(Z) \\ &\quad = \beta m\omega^2 \sum_i x_i^2 + \frac{\beta k}{N^2}\sum_{i,j} x_i x_j - \frac{2\beta k a}{N}\sum_i x_i - \beta k a^2 + 2\ln(Z) \\ &\quad = \beta\left(\frac{k+N^2 m\omega^2}{N^2}\right)\sum_i x_i^2 + \frac{\beta k}{N^2}\sum_{i\neq j} x_i x_j - \frac{2\beta k a}{N}\sum_i x_i - \beta k a^2 + 2\ln(Z) \end{aligned} \tag{2.15}$$

$$\begin{aligned} &\left(\vec{x}-\vec{\mu}\right)^T \boldsymbol{S}\left(\vec{x}-\vec{\mu}\right) + 2\ln(A) \\ &\quad = \sum_{i,j}\left[S_{ij}x_i x_j - S_{ij}\mu_i x_j - S_{ij}x_i\mu_j + S_{ij}\mu_i\mu_j\right] + 2\ln(A) \\ &\quad = \sum_i S_{ii}x_i x_i + \sum_{i\neq j} S_{ij}x_i x_j - 2\sum_{i,j} S_{ij}x_i\mu_j + \sum_{ij} S_{ij}\mu_i\mu_j + 2\ln(A) \\ &\quad = S_{11}\sum_i x_i^2 + S_{12}\sum_{i\neq j} x_i x_j - 2\mu\sum_{i,j} S_{ij}x_i + \mu^2\sum_{i,j} S_{ij} + 2\ln(A) \end{aligned} \tag{2.16}$$

A few simplifications have been made in the final expression for the MVN, based on the symmetry of the potential. On the third line of eq (2.16), the symmetric nature of $\boldsymbol{S}$ is used to combine the two terms linear in x from the line above. Since the potential is symmetric in the interchange of particles, so too is the covariance matrix and its inverse. The same assumption of interchangeability permits the substitution of S_{11} for any diagonal element, of S_{12} for all off–diagonal elements, and of μ for all μ_i. Looking at the first two terms in eqns (2.15) and (2.16), the elements of $\boldsymbol{S}$ are:

$$S_{ii} = \frac{\beta(k + N^2 m\omega^2)}{N^2} \qquad \text{and} \qquad S_{ij} = \frac{\beta k}{N^2} \ . \tag{2.17}$$

Given that $\boldsymbol{S}$ and the covariance matrix $\boldsymbol{C}$ are inverses, the covariances can be calculated by solving the equation:

$$\delta_{ij} = \sum_k S_{ik} C_{kj} \ . \tag{2.18}$$

$$\begin{aligned} 1 &= S_{ii} C_{ii} + \sum_{k \neq i} S_{ik} C_{ki} \\ &= \Big[S_{ii}\Big] C_{ii} + \Big[(N-1) S_{ij}\Big] C_{ij} \end{aligned} \tag{2.18.a}$$

$$\begin{aligned} 0 &= S_{ii} C_{ij} + S_{ij} C_{jj} + \sum_{k \neq i,j} S_{ik} C_{kj} \\ &= \Big[S_{ij}\Big] C_{ii} + \Big[S_{ii} + (N-2) S_{ij}\Big] C_{ij} \end{aligned} \tag{2.18.b}$$

Eqs (2.18.a) and (2.18.b) constitute a linear system of two variables whose solution is:

$$\begin{aligned} \text{Variance: } C_{ii} &= \frac{1}{\beta m\omega^2} \left(1 - \frac{k}{N(k + Nm\omega^2)}\right) \\ \text{Covariance: } C_{ij} &= -\frac{1}{\beta m\omega^2} \frac{k}{N(k + Nm\omega^2)} \end{aligned}$$

In addition to the covariance matrix, the mean of each particle can found by comparison of the linear terms in eqns (2.15) and (2.16).

$$\begin{aligned}\frac{\beta k a}{N}\sum_i x_i &= \mu\sum_{i,j} S_{ij}x_i = \mu\sum_i \Big(S_{ii} + (N-1)S_{ij}\Big)x_i \\ &= \mu\left(\frac{\beta(k+N^2m\omega^2)}{N^2} + \frac{\beta(N-1)k}{N^2}\right)\sum_i x_i \\ &= \mu\frac{\beta(k+Nm\omega^2)}{N}\sum_i x_i\end{aligned}$$

$$\mu = \frac{ka}{k+Nm\omega^2} \tag{2.19}$$

Summarizing the results above, the biased harmonic particles of eq (2.12) form an ensemble equivalent to the MVN distribution described below.

$$P(\vec{x}) \equiv MVN_{\vec{\mu},\boldsymbol{C}}(\vec{x}) \qquad \begin{aligned}\mu_i &= \frac{ka}{k+Nm\omega^2} \\ C_{ii} &= \frac{1}{\beta m\omega^2}\left(1 - \frac{k}{N(k+Nm\omega^2)}\right) \\ C_{ij} &= -\frac{1}{\beta m\omega^2}\frac{k}{N(k+Nm\omega^2)}\end{aligned} \tag{2.20}$$

If k is increased toward infinity, the mean approaches a. In this case, $N = 2$ is sufficient for the restraint. But in systems other than this example, N may need to be very large. If N tends toward infinity, k must grow faster than N. The variance of any particle will tend toward the unbiased value as the biasing constant and number of samples increase. The growth of N is key in this situation, unlike for the mean. Additionally, the covariance between any two particles will vanish in this case, tending toward a decoupling of the particles as in the unbiased case.

While this example suggests that this method of biasing adjusts the observable of interest into accordance with experiment while allowing other properties to remain at their unbiased values, this is not an exhaustive proof. It does however, illustrate why restraining the average value of a property is

useful. The following section outlines the behavior a biasing potential should have in order to maximize agreement between quantified properties of the experiment while minimizing the impact on the probability distribution of the system over configuration space.

2.1.3 Perturbing The Free Distribution Using Restrained Ensemble Sampling

Roux and Weare demonstrate how using a restrained ensemble affects the resulting distribution of configurations in a simulation.[89] Their results are expanded upon after applying the method to a similar Monte Carlo sampling of a particle in a one dimensional potential. The potential used (dimensionalized by k_BT) is given by

$$\beta U_0(q) = 25(q-0.25)^4 - q\cos(q) + \frac{\sin(20q)}{q^2+0.5}\ . \tag{2.21}$$

The potential will be sampled in the canonical ensemble, yielding the following probability distribution for the particle.

$$P(q) = \frac{e^{-\beta U_0}}{Z} = \frac{1}{Z}\mathrm{EXP}\left(-25(q-0.25)^4 + q\cos(q) - \frac{\sin(20q)}{q^2+0.5}\right) \tag{2.22}$$

$Z = \int dq\ e^{-\beta U_0} \approx 1.87628$ is the partition function. The potential and resulting probability density function are presented in figure 2.1. The normalization can be absorbed into the exponential, making the partition function equal to one, and effectively shifting the potential by a constant. Shifting the energy by a constant does not affect the dynamics, which is in agreement with the equivalence of representation of the probability distribution.

$$\beta U_0(q) = 25(q-0.25)^4 - q\cos(q) + \frac{sin(20q)}{q^2+0.5} + 0.62929 \tag{2.21$'$}$$

$$P(q) = \mathrm{EXP}\left(-25(q-0.25)^4 + q\cos(q) - \frac{sin(20q)}{q^2+0.5} - 0.62929\right) \tag{2.22$'$}$$

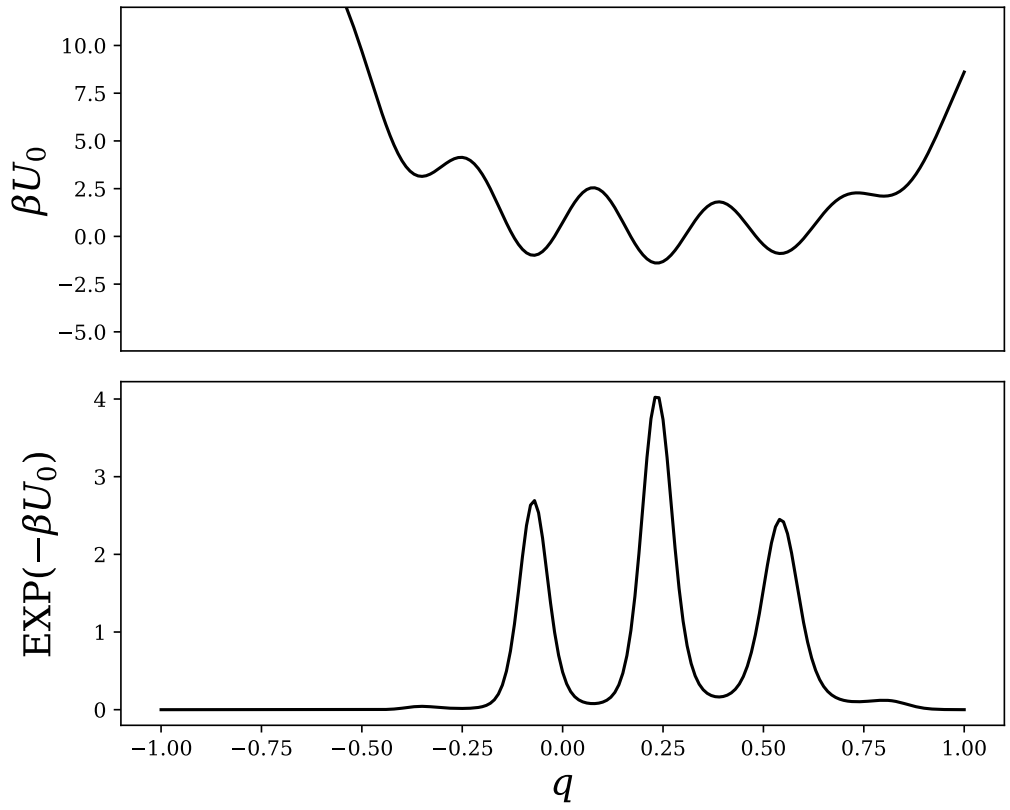

Figure 2.1: One dimensional potential and probability density function for Monte Carlo simulations equivalent to those used by Roux and Weare.[89] The density plotted represents the distribution to be sampled in the absence of restraint. It has positive expectation value $Q_0 = 0.258$ before restraint. The density modified by using the restrained ensemble will have expectation value $Q = -0.127$.

A simple Monte Carlo simulation, using the Metropolis-Hastings algorithm to sample, was constructed in Python (see appendix B.3.1) using acceptance criteria based on eq (2.22). First, sampling with no restraining was performed to verify that the algorithm could reproduce the probability density used as input (results in figure 2.2).

To perform a restrained ensemble simulation, N replicas of the particle were subjected to the same algorithm as the free particle except for an im-

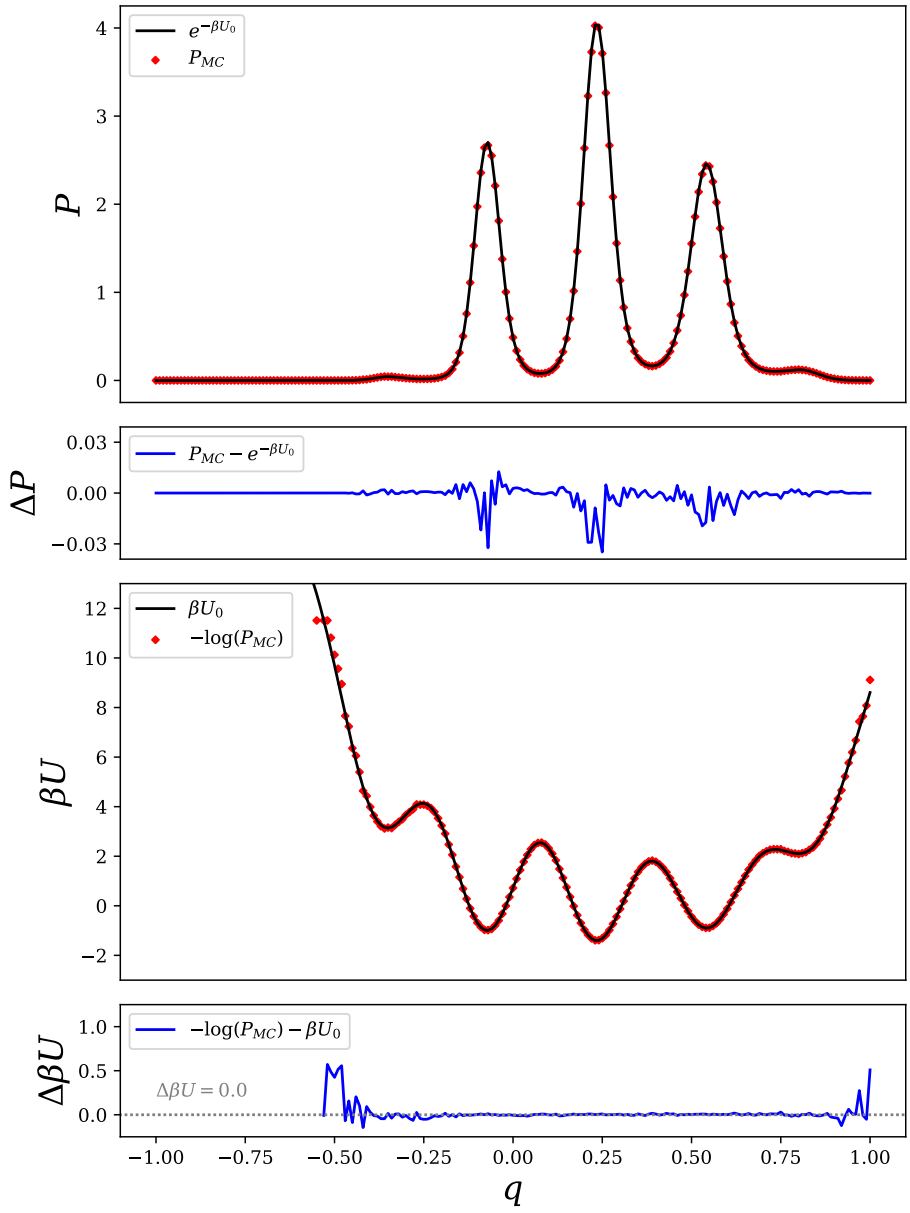

Figure 2.2: Analytic (U_0) and Monte Carlo reproduced (MC) probability density and potential for eqs 2.21 and 2.22. The MC produced density comes from the single replica, unrestrained simulation with 10^7 samples.

posed condition on the allowed movements of the system. The positions of the particles, q_i, have their average equal to some observed value, Q, at each step in the simulation.

$$\frac{1}{N}\sum_i q_i = Q \tag{2.23}$$

This condition acts as a bias on the ensemble average as proposed in eq (2.12) with the force constant taken to be infinite, *i.e.* a delta distribution on the average position of the ensemble. The details of how a constraint on the proposal distribution was accomplished can be found in appendix B.3. The unbiased average position in the potential is $Q_0 = 0.258$ and the bias restraint is $Q = -0.127$. The resulting probability density functions are show in figure 2.3 for simulations with different numbers of replicas.

The resulting distribution from two replicas is conspicuously symmetric, arising from the nature of restraining two replicas in this way. Because the average position must be conserved, any step away from the average by one particle is exactly matched by the other. This can also be seen in the covariance matrix for these particles positions, see eq (2.24).

$$\begin{aligned}\mathrm{Cov}(N=2) &= \begin{pmatrix} \left\langle (q_1 - Q)^2 \right\rangle & \left\langle (q_1 - Q)(q_2 - Q) \right\rangle \\ \left\langle (q_2 - Q)(q_1 - Q) \right\rangle & \left\langle (q_2 - Q)^2 \right\rangle \end{pmatrix} \\ &= \begin{pmatrix} 0.00339 & -0.00339 \\ -0.00339 & 0.00339 \end{pmatrix}\end{aligned} \tag{2.24}$$

In defining the covariance, the substitution $Q = \langle q_1 \rangle = \langle q_2 \rangle$ was made due to the high level of accuracy seen in the numerical results. As is shown by Roux and Weare,[89] in the limiting case of infinite replicas, the covariances will vanish and all particles will have identical, independent probability distributions. Even in the finite case, the probability distributions would be the same for each particle (but not independent). This is because the interaction that governs their motion is symmetric under exchange of the particles. However, sampling issues prevent the realization of these distributions. Because the dimensionality of the system becomes large, it may take a very long time for the sampler to propose steps that move all particles to all positions in space as the number of replicas is increased.

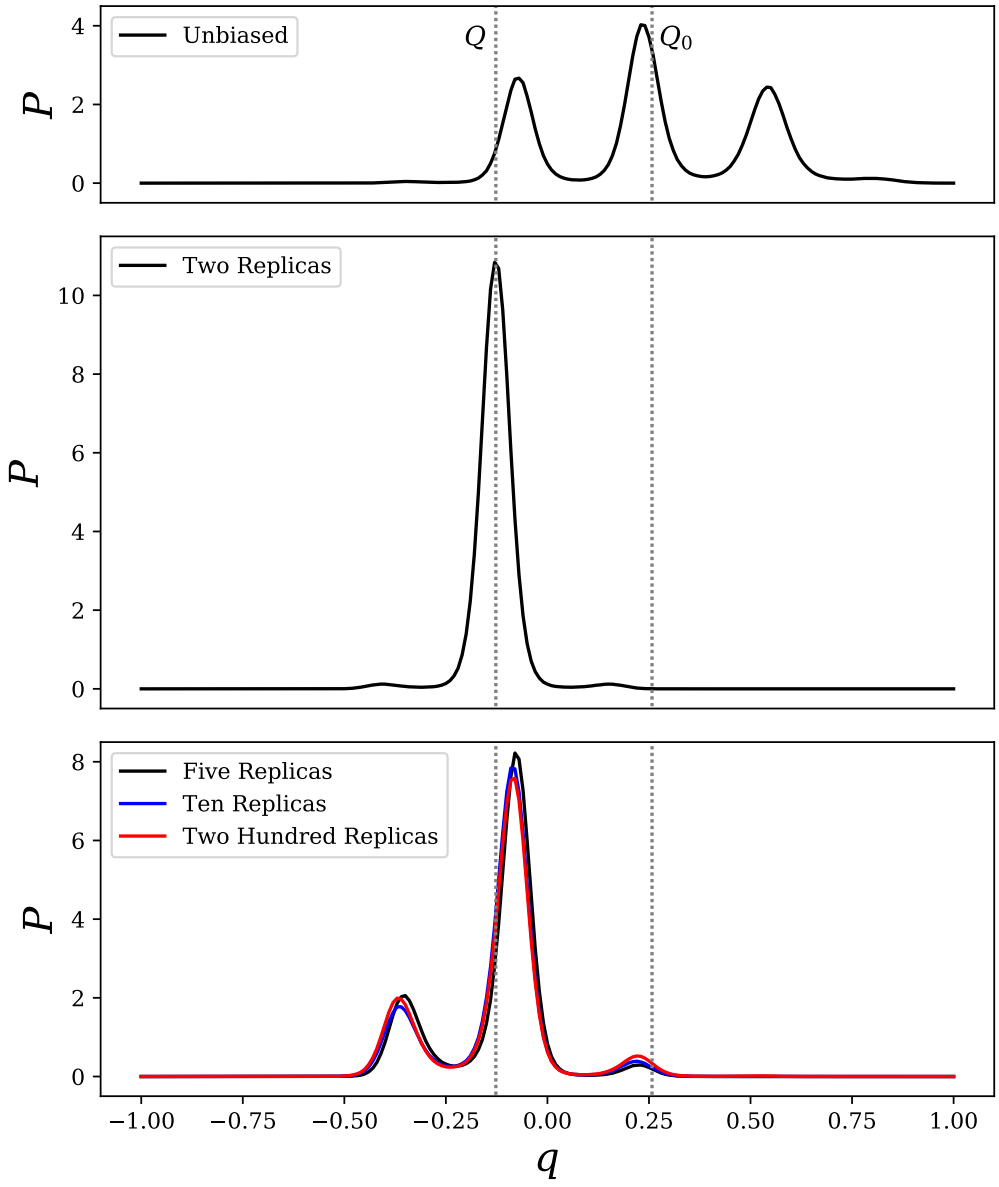

Figure 2.3: Unrestrained (top panel) and restrained probability density functions for the potential of eq 2.21. The densities quickly approach an asymptotic distribution as number of replicas increase. Ten and two hundred replicas are quite comparable.

Instead of examining an individual particle's density function, the entire ensemble is considered. If the replicas are well distributed over the domain in their initial positions, then the acceptance criteria of the sampler should correctly produce the probability distribution (given a large enough sample set). To verify the robustness of the distribution sampled, multiple runs with distinct initial conditions were checked for compatibility in their resulting distributions. Because of the difficulty in sampling the full distribution within a single replica, covariances are not considered a significant empirical measure in this analysis of convergence. The condition imposed on proposal steps in eq (2.23) creates a covariance that decreases with more replicas. However only very long sampling, with total exchange of particle positions, will permit an accurate measure of the covariance. Another measure on the probability distribution will be introduced later.

As illustrated in the bottom panel of figure 2.3, the main features of the probability density function for the ensemble of particles are quickly established as the number of replicas increase. Not much changes about the distribution between 5 and 200 replicas, though the distributions look significantly different from the unbiased simulation. How these changes arise is not immediately obvious from the probability density function, but can be understood using the effective potential that arises from this bias. To calculate the effective potential, one can invert eq (2.22′).

$$\beta U_P = -\log(P) = \beta U_0 + \beta U_{bias} \tag{2.25}$$

The effective potentials for simulations with different numbers of replicas are shown in the top panel of figure 2.4. Their differences of the effective potentials with the potential of eq (2.21) are shown in the bottom panel. The effective potential for $N = 2$ is relatively narrow, as compared to the others, which is in agreement with the strongly peaked density in figure 2.3. It should be noted that the restrained ensemble with one replica, which was not performed, would simply produce a Dirac delta distribution for its probability density function, as per the restraint in eq (2.23). As the number of replicas is increased, the effective potential becomes less restrictive and approaches a shape more similar to the unbiased potential.

The difference between the effective potentials and the unbiased potential is shown in the lower panel of figure 2.4. These differences represent the bias for each simulation that is added by restraining the ensemble, βU_{bias}. For low numbers of replicas, the added bias is not analytically known, but as the

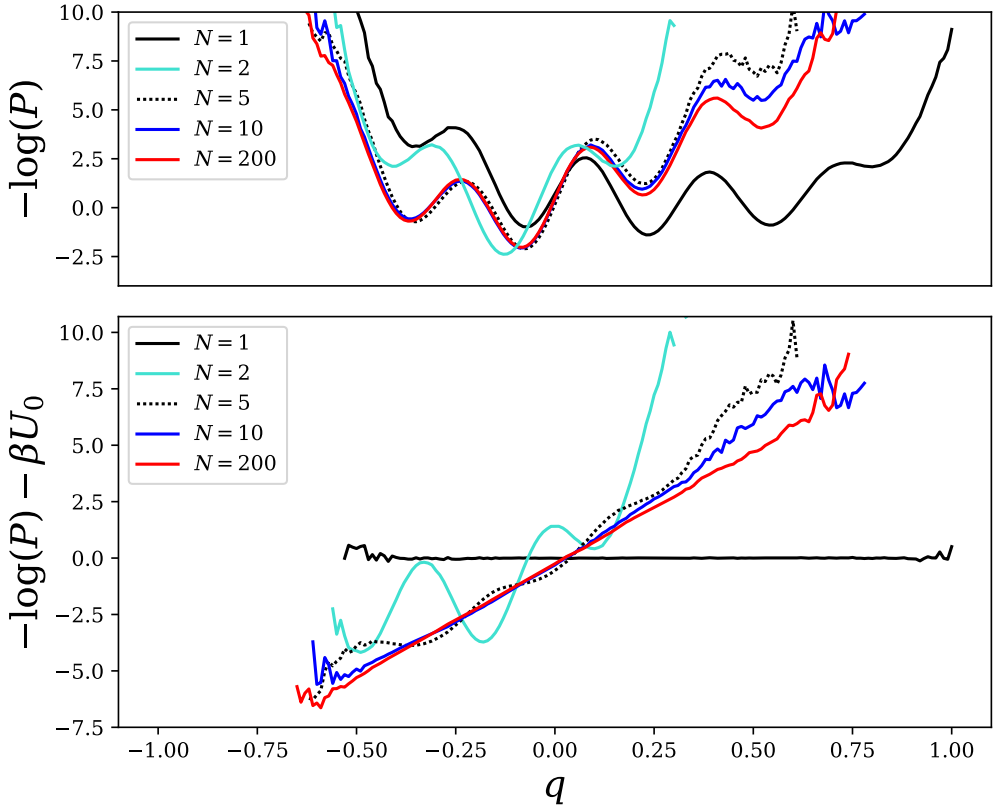

Figure 2.4: (Top panel) Effective potential defined in eq (2.25) derived from Monte Carlo produced density. (Bottom panel) Difference between effective potential and unbiased potential, describing the bias added by the restrained ensemble. As the number of replicas increase, the difference approaches a line. The line of best fit for 200 replicas is $\beta(\Delta U) = 10q$.

number of replicas increase a pattern emerges. For $N = 200$ replicas, the range of q with a significant number of samples ($-0.5 < q < 0.5$) is fit well by a line through the origin with a slope of ten.

Roux and Weare present a thorough theoretical treatment of how the probability density function changes and what effective potential is created using restrained ensemble sampling;[89] the results are summarized here. The main premise is that an effective bias will modify the probability distribution,

with respect to configuration space, in a way which is minimally invasive while still achieving agreement between the expectation values of observables calculated from simulation and those measured in experiment. The measure of similarity between the original and perturbed probability densities is the Kullback-Leibler (KL) divergence functional. The KL divergence for the original probability density, P_0, and some other distribution, P, is defined as:

$$D_{KL}\left[P(\vec{X})||P(\vec{X})\right] = \int d\vec{X}\ \ P(\vec{X}) \log\left[P(\vec{X})/P_0(\vec{X})\right] \tag{2.26}$$

$D_{KL} \geq 0$, as shown in appendix B.1, being minimal when the two distributions are equal. The goal of a minimally invasive biasing procedure is to find the probability distribution that minimizes the KL divergence and satisfies

$$Q = \int d\vec{X}\ \ q(\vec{X})P(\vec{X})\ , \tag{2.27}$$

which describes the agreement with experiment. Additionally, the distribution should be normalized.

$$1 = \int d\vec{X}\ \ P(\vec{X}) \tag{2.28}$$

Roux and Weare point out that this is a problem of constrained optimization of D_{KL} and that the solution can be found using the method of Lagrange multipliers.[89] Performing a constrained variation of the KL divergence gives an equation to solve for the desired distribution.

$$\begin{aligned} 0 = \frac{\delta}{\delta P(\vec{X})}\Bigg[&\left(\int d\vec{X}\ \ P(\vec{X}) \log\left[P(\vec{X})/P_0(\vec{X})\right]\right) + \alpha\left(1 - \int d\vec{X}\ \ P(\vec{X})\right) \\ &+ \lambda\left(Q - \int d\vec{X}\ q(\vec{X})P(\vec{X})\right)\Bigg] \\ = \log\left[P(\vec{X})\right] - \log\left[P_0(\vec{X})\right] &+ 1 - \alpha - \lambda q(\vec{X}) \end{aligned} \tag{2.29}$$

This equation can be inverted for the desired probability density,

$$P_*(\vec{X}) = P_0(\vec{X})e^{\alpha-1}e^{\lambda q(\vec{X})} \tag{2.30}$$

As was the intent through its addition as a Lagrange multiplier, the term containing α reconciles the normalization of the new distribution with the old. With a suitable definition of P_*, this term is easily determined and need not be considered further. The term associated with the experimental observable appears linearly in q within the exponential. This is exactly the behavior in the case under study. Rewriting the effective potential using a linear term as the bias, shows the equivalence of the two probability distributions, as suggested by the bottom panel of figure 2.4.

$$\beta U_P = \beta U_0 + mq \tag{2.31}$$

$$P_* = \frac{e^{-\beta U_P}}{\int dq\ e^{-\beta U_P}} = \frac{P_0\ e^{-mq}}{\int dq\ P_0\ e^{-mq}} \tag{2.32}$$

The numerator of the term on the right of eq (2.32) illustrates how the method of restrained ensemble, in our example, inherently finds the probability distribution that minimizes the KL divergence while matching observable expectations. No prior calculation of the Lagrange multiplier weight is necessary. The earlier fit value of $m = 10$ gives $\langle q \rangle = -0.127119$, reasonably close to the target Q.

Ideally, an infinite number of replicas is desired for a restrained ensemble simulation, but computational cost limits the number attainable. In order to choose the fewest replicas necessary to approximate similarity to the distribution presented in eq (2.30), it is instructive to plot the KL divergence as a function of number of replicas in the restrained ensemble. These results for the potential in eq (2.21) are shown in figure 2.5.

A simulation for $N = 1$ was not performed because it would simply result in a delta distribution with the particle fixed. The KL divergence in this case is positive infinity. $N \geq 2$ produces a finite value and by $N = 10$, the KL divergence is within 0.3% of the theoretical value for the distribution in eq (2.30) with $\lambda = -10$. The computational expense of further increasing the number of replicas does not provide much improvement in the quality of the probability distribution. Furthermore, the KL divergence appears to increase after 50 replicas, but this is an undersampling issue. In order to verify, the simulation for $N = 200$ was performed with ten times as many samples. The higher sampling reestablished agreement with the theoretical prediction. This suggests that the increase in number of replicas requires more simulation to provide the same accuracy in the probability density function. Therefore,

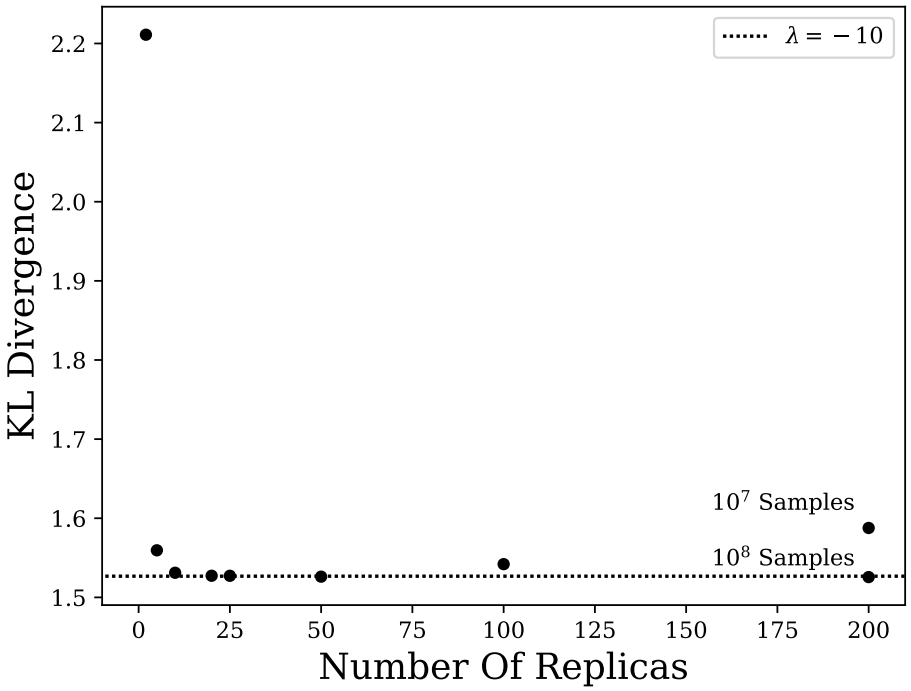

Figure 2.5: Kullback-Leibler divergence for histogrammed distributions of restrained ensemble simulations with $N = \{2, 5, 10, 20, 25, 50, 100, 200\}$ replicas. The KL divergence quickly converges toward the theoretical value for the distribution in eq (2.30) with $\lambda = -10$ (dotted line). All points for $N < 200$ were performed with 10^7 samples in total, leading to an apparent increase of the KL divergence (from undersampling). $N = 200$ was performed again with 10^8 samples, restoring agreement with the theoretical value.

the number of replicas must be chosen to optimize the minimization of the KL divergence while maintaining a sufficiently low sample number to accurately capture the resulting probability density.

2.2 Steering Of Simulations Using Neutron Data

The following few sections will discuss the use of the component occupancy volume profile, discussed in the introduction, as an observable to bias within a simulation context. First, preparation and construction of the protein CVO profile is discussed. Then, a bias potential is constructed using the resulting density profile. The restrained ensemble approach discussed in the previous sections is too costly to implement but the use of restraint on time-averaged quantities is explored as a viable alternative. Finally, provided is a discussion of technical obstacles in the implementation and how they were addressed.

2.2.1 The Observable Used In Biasing From Neutron Reflection

In the biasing of simulations based on NR data, the CVO profile of a protein provides an observable to match in simulation. It is constructed from the z-coordinates of the protein atoms alone, and is therefore easily computed throughout the course of simulation. In preparation for constructing a biasing potential, it is useful to note that the CVO profile intrinsically has information about the surface coverage of the protein on the membrane. To see this, take the integral of the CVO profile, ϱ, over all z. This is proportional to the volume of the protein, since the CVO profile represents the amount of volume occupied at a given point in z.

$$\int dz \ \varrho_{pr} \propto V_{pr} \tag{2.33}$$

The CVO is a unitless fraction, its integral has dimensions of length. This means there is a standard of area relating this integral to the volume. The CVO profile represents a material density along the z-axis and scaling it by an area gives it the units of linear density. The area standard of choice is the area per lipid, since the lipids fill the volume in their tails and a CVO profile scaled by this area provides easy calculation of protein surface coverage on the membrane – a quantity often reported in protein-membrane experiments. In preparing the simulation, the x-y area of the simulated (periodic) box determines the number of lipids of the system (with z as the interfacial normal) and, therefore, the ratio of the protein volume to the

lipidic volume. The box size should be chosen to have this ratio be close to the ratio of the integrated CVO profiles.

$$\frac{V_{pr}}{V_{lip}} = \frac{\int dz\ \varrho_{pr}}{\int dz\ \varrho_{lip}} \tag{2.34}$$

In practical terms, it can be difficult to match this condition exactly but small variations in surface concentration may not significantly alter the protein behavior. Fortunately, what we seek to do is correct for deviations from experimentally observed ensembles, so small changes in surface concentration should not be an issue.

In order to standardize the potential, it is preferable to avoid case by case calculations of relative areas to bring the simulation CVO profile into agreement with the experiment. Instead, the profile is normalized such that the integrated density is one. The new density, ρ, will be the one used in biasing the simulation.

$$\rho(z) = \frac{\varrho(z)}{\int dz\ \varrho(z)} \tag{2.35}$$

ρ, when multiplied by the volume of the protein it corresponds to, represents the average profile for all proteins in either the experiment or the simulation. In calculating ρ for simulation, each atom's density is represented by a normal distribution and the total distribution scale by the number of atoms to ensure it integrates to unity.

$$\rho_{sim}(z) = \frac{1}{N}\sum_{i=1}^{N} \mathcal{N}\left(z|z_i, \sigma\right) \tag{2.36}$$

In the simulations that follow, $\sigma = 1\ \mathring{A}$. To remove unnecessary computational expense, each atom's distribution is truncated at $\pm 3\sigma$ as density contributions beyond this contribute about a quarter of a percent to the area of the profile.

2.2.2 The Bias Potential

Having constructed ρ_{sim} and taking ρ_{exp} to be an experimentally determined CVO profile normalized using eq (2.35), a biasing potential is added to the

simulation in the form

$$U_{NR}(z) = \lambda \left(\rho_{sim} - \rho_{exp}\right) \tag{2.37}$$

A potential of this form is akin to a Lagrange multiplier, vanishing when the two densities are in agreement. A Lagrange multiplier might vary as a function of z and treating each point in the density as a separate observable to restrain may also require z dependence for λ. For the following, however, λ will be held constant in space as a first approximation. In GROMACS, the energy units are $\frac{kJ}{mol}$ and units of length are measured in nm. This gives ρ units of nm^{-1} and λ units of $\frac{kJ \cdot nm}{mol}$ in this implementation.

Figure 2.6 shows how the density profiles from simulation and experiment are combined to form the potential of eq (2.37). As an example, a helical peptide consisting of 35 alanine residues, which was simulated as a toy model. The results of those simulations follow in subsequent sections.

Panel A depicts an example target (experimental) density for the helix, describing a system that prefers to orient the peptide along the z-axis. Panel B depicts a configuration of the simulation in which the helix does not completely align along the axis of orientation intended by the target. Panel C shows the difference between these two densities, which when scaled by λ is the bias potential of eq (2.37). In a region along z where the density produced in simulation is larger than that observed in experiment, the bias potential is larger and penalizes the extra density by repelling atoms from that region. A region where the experiment has more density will have a smaller potential and therefore attracts atoms toward that region. Panel D illustrates the effect of this potential on the configuration depicted in panel B. The regions where the potential has the largest gradients are the regions with the strongest forces. Therefore, in the current example, the ends of the helical peptide receive a torque inducing force that acts to align protein along the desired axis.

The appropriate value of λ to use will vary depending on the system under study. When using the method of restrained ensemble, the choice of λ is implicitly selected through the way the replicas are coupled. Unfortunately, there is a large cost to running multiple replicas of a simulation when also trying to maximize the complexity and size of the system under study. As an alternative, the history of the densities visited along the trajectory can be used to incorporate information from a larger set of conformations. Therefore, averaging the densities observed throughout the simulation is explored

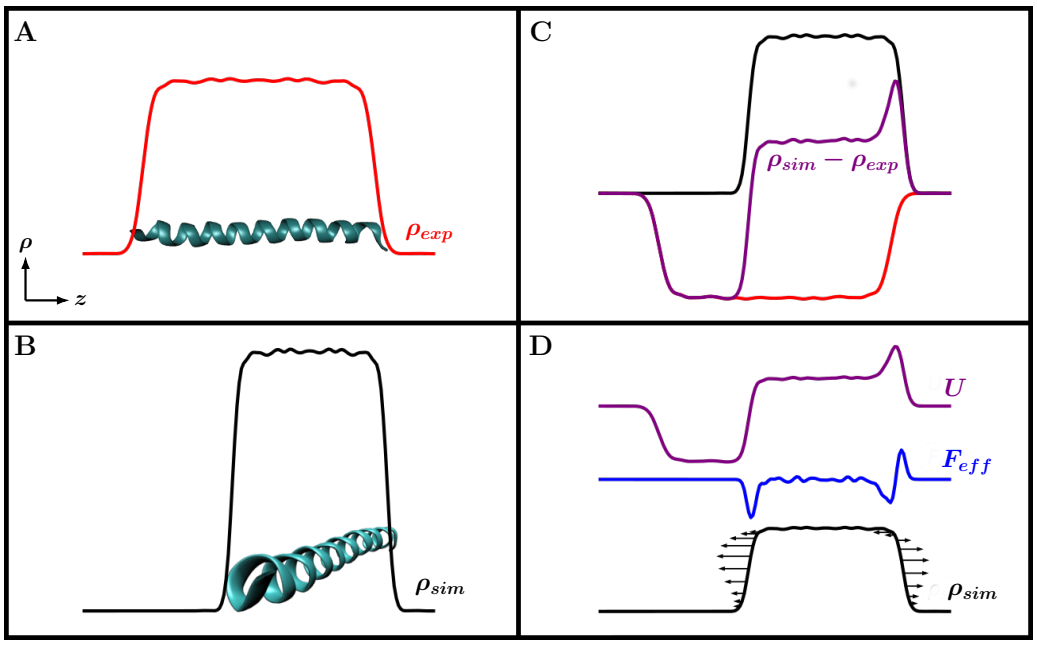

Figure 2.6: Construction of the bias potential using a helical peptide. Panel A depicts the experimental density, with the peptide oriented along the z-axis. Panel B shows the same peptide oriented off-axis in a simulation and its resulting density. Panel C shows the difference of the experimental and simulation densities, which is proportional to the potential. Panel D shows the bias potential and the resulting forces added to the simulation. The black simulation density includes arrows to illustrate what portions of the protein's profile are affected by the forces.

as a mode of biasing. The time averaged density is

$$\langle \rho_{sim} \rangle_t = \frac{1}{M} \sum_{i=1}^{M} \rho_{sim}(t_0 + i\tau) \tag{2.38}$$

where τ is the interval of time between sampling of the configuration for the average density. The integrator will need to undergo many time steps before the configuration of the system is distinct enough to meaningfully impact the average density. The fastest side chain and backbone motions have a time scale on the order of picoseconds.[90,91] For this reason, sampling

to update $\langle \rho_{sim} \rangle_t$ in the **historied bias** and ρ_{sim} in the **memoryless bias** occurs every 0.1 ps in the current implementation. The differences between using a historied bias and a memoryless bias is explored in the next few sections.

2.2.3 Handling Large Discrepancies Between Densities

Because the forces on the protein correlate with gradients in the potential, large amounts of spatial separation between the centers of the experimental and simulation profile can lead to large forces at the edge of the protein. For example, imagine that, during simulation, the peptide in figure 2.6 barely overlaps with the experimental profile on one side. The net effect of the biasing force on the peptide is to pull the protein to bring the centers into alignment. The forces applied to move the whole protein are localized on a the small portion of atoms that make up the edge of the density overlap. This could, and did in short tests, strongly distort the secondary structure.

It may be necessary to start the protein in a configuration that is spatially disjoint from the target profile, as in the case of docking a protein with a membrane. A way to manage these cases is to treat the central position of the experimental profile as a restrained observable, independent of the shape of the density (relative to its center). In the current implementation, this restraint is applied harmonically but would benefit from a bias like that suggested in section 2.1. The harmonic restraint applied applies a uniform force to each atom.

The application of restraint to the central position is defined through the first and second moments of the profiles.

$$\mu^{(1)} = \int dz \; z \, \rho(z) \tag{2.39}$$

$$\mu^{(2)} = \int dz \; \left(z - \mu^{(1)}\right)^2 \rho(z) \tag{2.40}$$

The first moment is a measurement of the center of the profile. $\mu^{(1)}_{sim}$ and $\mu^{(1)}_{exp}$, the centers of the simulation and experimental profiles, are considered significantly separated when their distance is larger than the square root of the experimental profile's second moment, which is a measurement of the

profile's width. When

$$\left|\mu_{sim}^{(1)} - \mu_{exp}^{(1)}\right| > \sqrt{\mu_{exp}^{(2)}} \tag{2.41}$$

the bias between the profile shape and position is decoupled. ρ_{exp} is shifted along z by the difference in the profile centers before incorporation into the potential, biasing the system as if the profile centers matched. Additionally, the aforementioned harmonic restraint is applied to every particle to bias the center toward its target location.

$$F_{cen} = \frac{k}{N}\left(\mu_{exp}^{(1)} - \mu_{sim}^{(1)}\right) \tag{2.42}$$

Note that eq (2.42) is the force, not the energy, and that the force constant shown is scaled by the number of particles, N. The resulting sum of all forces is a harmonic potential between the profile centers, using the entire mass of the protein to determine the harmonic constant's strength.

In the simulations that follow, k was chosen such that there is a smooth transition when eq (2.41) signals to turn off the bias of the center – eq (2.42). Specifically, the force per atom near the transition is similar to the force per atom from eq (2.37) on the boundary of mismatch between the profiles. This is not completely rigorous and, as mentioned at the start of the section, future implementations could explore completely decoupling the restraint of the centers and the restraint on the matching of profiles, offsetting the z-axis of the experimental to align the center positions profiles. Since the mechanisms necessary are already implemented for the spatial mismatch condition, there is a framework for these changes in place.

2.3 Poly-Alanine (pALA) Simulations

2.3.1 Description

A 35 residue helical polyalanine peptide was constructed using Visual Molecular Dynamics (VMD) software by chaining together a smaller seven residue helical unit. The peptide was capped with an acetyl and an amide group, solvated with 100 mM NaCl in SPC/E water (CHARMM 36, July 2017 forcefield), and put in a cubic box with side length 8.83 *nm*. The system went

through 50000 energy minimization steps, equilibration at fixed volume/temperature and fixed pressure/temperature for 100 ps each, and simulated in a production run for 10 ns ($T = 300\text{K}$, $P = 1\text{bar}$).

After simulation, the 10 ns trajectory was analyzed to produce a mock density profile. To do this, the frames were aligned such that the protein's center of geometry was stationary and also rotated such that the principal axes of the protein's moment of inertia tensor remained fixed. An average protein density was calculated using the centered and aligned trajectory, as per eq (2.36). The profile obtained is pictured in Figure 2.7.

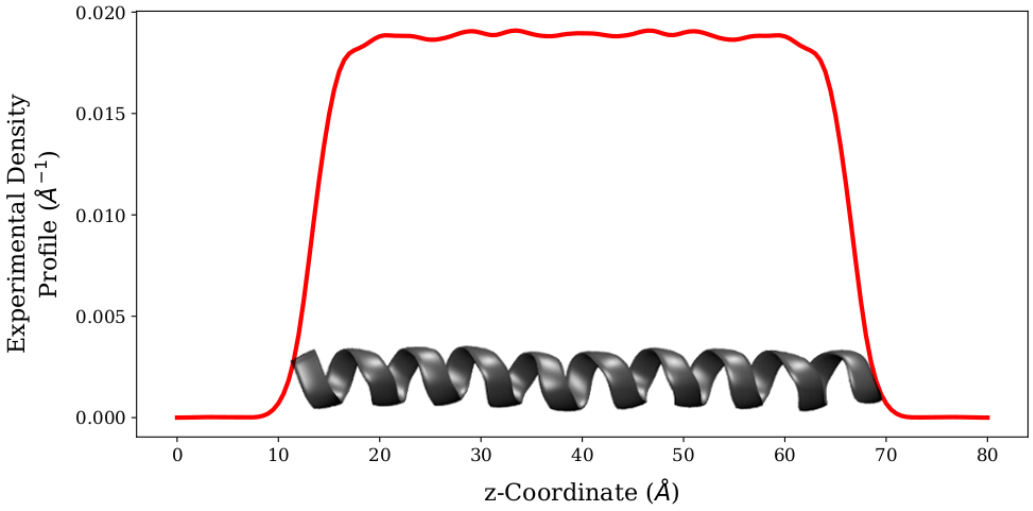

Figure 2.7: Mock density profile for helix of alanine. The experimental density calculated from the average taken in the initial simulation is shown in red. A typical configuration contributing to the average is overlaid in gray.

The mock profile, representative of a rigidly oriented and spatially constrained helix, acts as a substitute for an NR observable. This profile is a model for a peptide in a severely restricted configuration space as compared to the free case; biasing toward this profile would be resisted entropically. An experimental density would likely be much more similar to that of a free simulation than this mock profile, but this test case will serve to demonstrate strengths and weaknesses of the method. Simulations at two values of bias strength, $\lambda = \{0.05, 0.5\}\, k_B T \cdot nm$, were performed using both the memoryless and historied bias. An unbiased simulation was also performed

for comparison.

2.3.2 Analysis Of Simulation Results

Various observables were calculated for the five simulations and are shown in figures 2.8 - 2.11. Figure 2.8 shows the position of the profile's center ($\mu^{(1)}$) over the course of each simulation. The first two panels show the stronger of the two biases, the first using the memoryless bias and the second using the historied bias. The next two panels are the weaker bias in the same order; the last panel is the simulation without bias. All five panels have a histogram of the time series at the right of the plot.

Both simulations using memoryless bias fluctuate less, in their profile's center position (figure 2.8), about the average than their respective counterpart using a historied bias. The unbiased simulation appears to have its central positions distributed uniformly, if more statistics were available to smooth the fluctuations in the distribution. The historied simulation with $\lambda = 0.5\ k_B T \cdot nm$ shows a distinct bimodality in its distribution. The main peak of the weaker, historied simulation is located off center in the histrogram and also away from the center of the target profile ($\mu^{(1)}$), suggesting bimodality could also be occuring in this simulation. A possible reason for this bimodality will be discussed later.

As observed with the center trajectories, the root mean square difference (RMSD) between the experimental and instantaneous profile of the simulation

$$RMSD = \left[\int dz\ \left(\rho_{sim} - \rho_{exp}\right)^2 \right]^{1/2} \tag{2.43}$$

has smaller fluctuations about the mean for the memoryless bias than for the historied bias simulation for both strengths used, as shown in figure 2.9). The distribution of values for RMSD of the unbiased simulation is more difficult to infer than for the center position. The histogram suggests that there is a peak in the distribution and that there is more weight in RMSDs below the peak than above. The biased simulations show a similar trend, although the aspect ratio of the histogram varies between simulations. The memoryless biases have distributions that are less broad and, additionally, the peak in the distribution is at a lower RMSD than for the historied bias (though only by a small amount for the weaker bias). Furthermore, the historied simulations

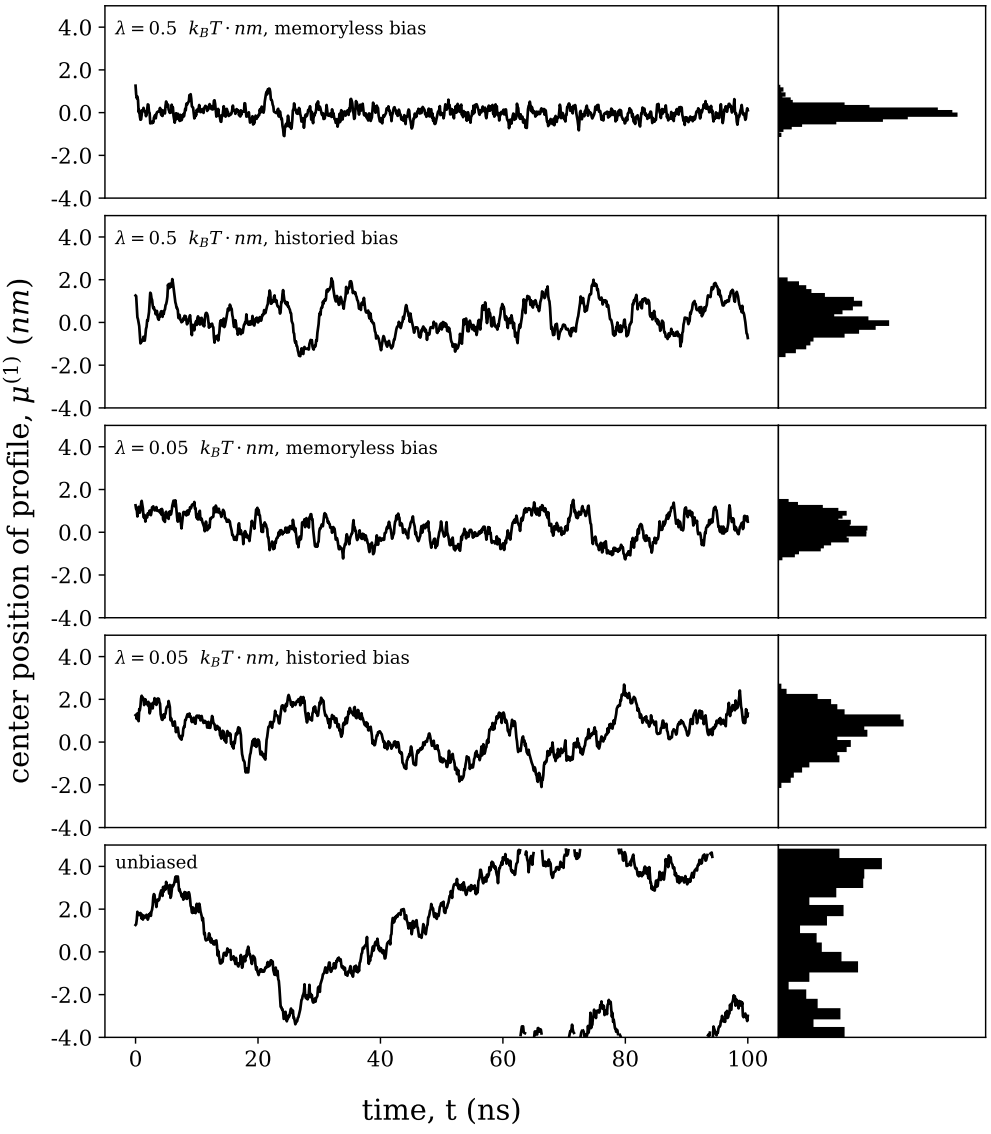

Figure 2.8: Central position, $\mu^{(1)}$, of the simuated pALA profile, as measured in eq (2.39). At the right of each time series, the values observed are collected in a histogram. The biased simulations produce center positions which are confined about the target profile center, the values in the unbiased simulation are distributed to all positions.

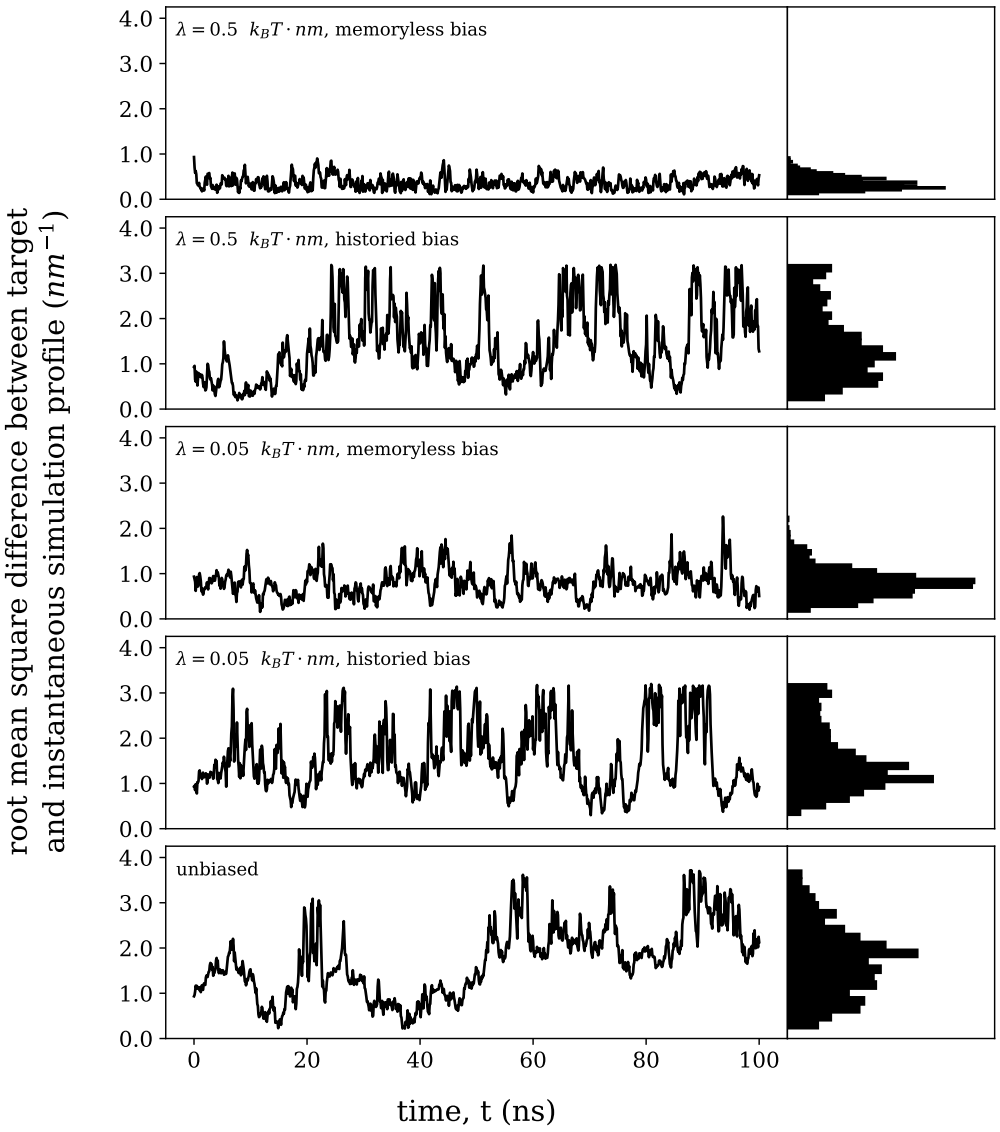

Figure 2.9: Root mean square difference between target profile (figure 2.7) and simulation profile calculated at the plotted time step. Histograms of the values in the time series shown at the right of each series. The unbiased simulation has a higher average and maximum RMSD value than any biased simulation. The memoryless biases have RMSD values that are smaller on average and have smaller maximum values as compared to the RMSD values observed in the historied biases.

have more weight at the highest RMSD values observed throughout all simulations.

As figures 2.8 and 2.9 show, the memoryless bias restricts fluctuations of $\mu^{(1)}$ and the RMSD more than the historied bias or unbiased simulations. To gain insight into the factors that contribute to these behaviors, a simple model for the system is introduced. To motivate this model, the root mean square fluctuations (RMSF) of each atom was calculated

$$RMSF_i = \left[\frac{1}{T} \sum_{t=1}^{T} \left(\vec{r}_i(t) - \langle \vec{r} \rangle \right)^2 \right]^{1/2} \tag{2.44}$$

where i indicates the i^{th} atom. The position of the atom, $\vec{r}_i(t)$, at timestep t is measured after the protein is centered and rotated to align with the initial configuration; $\langle \vec{r} \rangle$ is the time average of this quantity. All the atoms in the peptide deviated less from their intial position in the helix than in the unbiased simulation (see figure 2.10). The peptide is short helix and biasing did not affect this; it will be treated as a rigid rod that is diffusing in the solvent.

Treating the peptide as a rod, there are two major degrees of freedom that determine ρ_{sim}, the z-coordinate of the center of geometry and the angle of the helical axis relative to the z-axis. These affect the RMSD by changing the overlap of the two profiles and the relative height/width. As the center position moves along the z-axis, the decrease in overlap of ρ_{sim} and ρ_{exp} leads to an increase in RMSD. As the peptide rotates away from the z-axis, ρ_{sim} narrows and increases in height, and causes it to overfill ρ_{exp} in some regions and underfill in others (see figure 2.6).

The memoryless biases produced fluctuations of these degrees of freedom that were smaller than those for the historied biases. Figure 2.8 illustrates this for the center position degree of freedom. The weaker memoryless bias, however, produced a distribution that is almost as broad as the historied version despite the RMSDs attaining higher values and visiting the higher values more frequently (figure 2.9). The central position never deviates far enough in any of the biased simulations to be the main contribution to the RMSD. It is the orientation of the helix that plays the more significant role. The weaker memoryless bias did not permit orientational deviations larger than about $\pm 45^\circ$, and even less deviations were observed for the stronger bias. The historied bias, on the other hand, allows for complete reversal of orientation of the helix.The RMSD, however, is insensitive to reversal of the

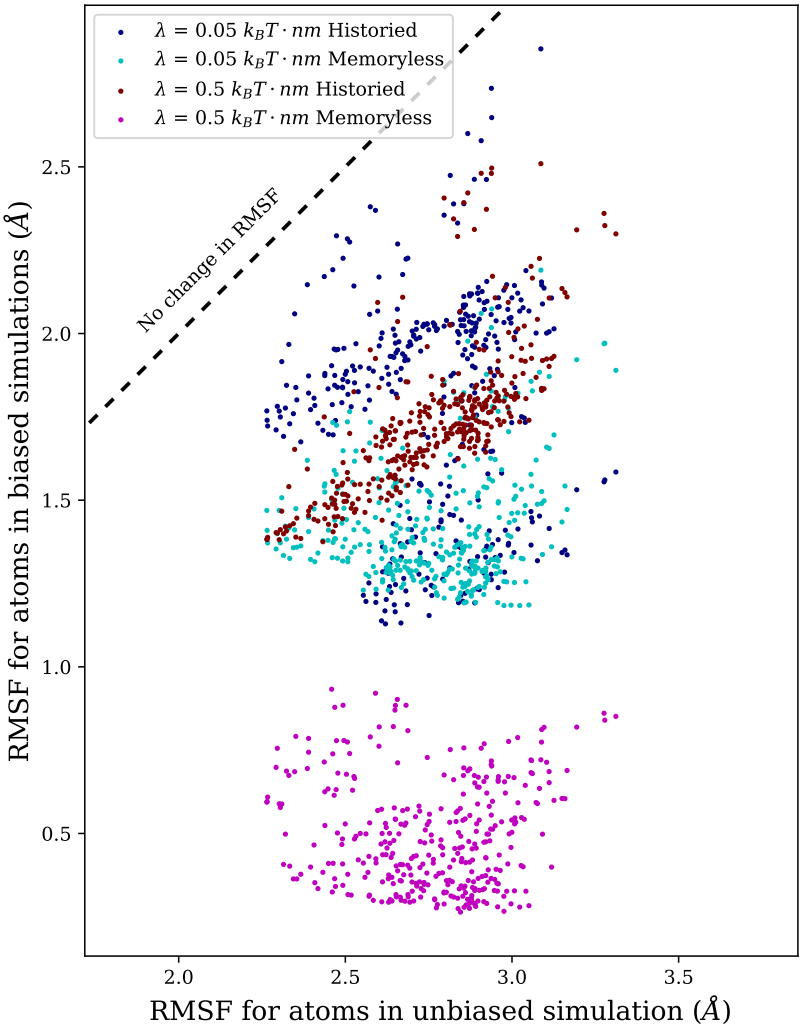

Figure 2.10: RMSF of atoms in biased simulations versus the RMSF of atoms in the unbiased simulation. All atoms in the biased simulations had RMSFs smaller than in the unbiased simulation (equality given by dashed line), resulting in structures which are typically more rigid in the biased simulation.

helix, due to the symmetry of the profile under negation of the polar angle of the helix. A scatter plot of RMSD versus $\cos\theta$ of the angle with the z-axis is shown in figure 2.11 and the symmetry is apparent in the shape of the data.

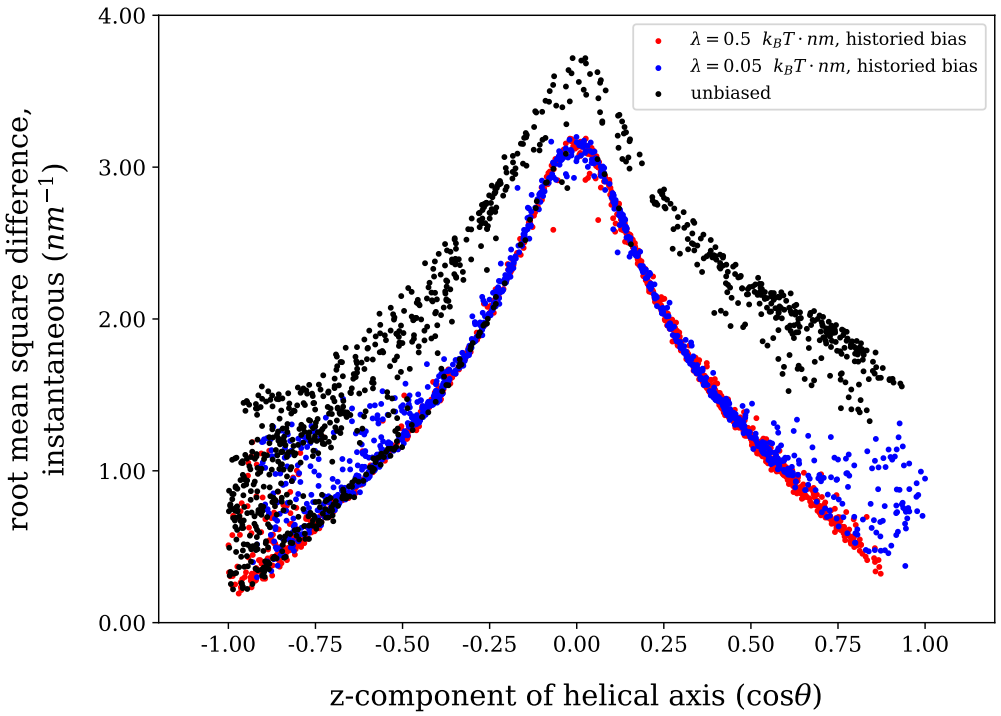

Figure 2.11: The RMSD between the target profile (figure 2.7) and the simulation profile versus orientation of the helix. The simulation density is calculated instantaneously at the plotted time step. The polar angle between the helical axis of the peptide and the z-axis is used in the orientation. The vertical range of RMSD values for fixed orientation primarily comes from variation in the center position of the profile, changing the amount of overlap with the target profile.

The same data is shown for the unbiased simulation and used to understand the pattern displayed by the data. The center of horizontal axis represents the helical axis pointing in some direction in the xy-plane ($\theta = 90°$). This corresponds to a narrow and tall ρ_{sim} with a large RMSD due to overfilling ρ_{exp} near the peptide and underfilling elsewhere. The vertical spread at a fixed angle is due to the variation in the center of the profile. In the unbiased simulation this occurs everywhere, including $\theta = 90°$ but not for the biased simulations because the center position is restricted by the bias

and the overlap of the profiles is always maintained in this orientation. This leads to less vertical spread in the data around the peak.

The vertical spread of the RMSD values toward $\theta = 0^\circ$ and 180° shows that the center position can vary enough to partially reduce the overlap in the densities. The amount to which this occurs is limited, since the unbiased simulation does not attain RMSD values as high as in the center of the plot. When oriented parallel to z, the periodic box prevents density mismatch as it is only about 160% the extent of ρ_{exp}. This puts a cap on the amount to which ρ_{sim} exist outside the envelope of ρ_{exp} before an oriented peptide traverses the periodic boundary and starts overlapping from the other side.

Figure 2.11 shows that, in the historied bias, moderate RMSDs (≈ 1.75) can arise for a parallel or antiparallel orientation in the historied bias. The largest RMSDs only occur in the historied bias when the orientations are far from aligned with z, where the $\mu^{(1)}$ fluctuations cannot contribute. These points illustrate why the RMSD histograms for historied simulations have a resemblence to both the histograms for the memoryless simulations and the unbiased simulation. The memoryless bias only fluctuates a small amount in both $\mu^{(1)}$ and θ, the modes that contribute to the RMSD. The RMSD histograms are fairly tight for these simulations. The unbiased simulation explores the entire range of RMSD values, but the values around 1.75 nm^{-1} have the largest range of orientations that can contribute these values. Therefore, the RMSD histogram is peaked around this value. The historied simulations explore all orientations but, for a range on either side of $\theta = 90^\circ$, $\mu^{(1)}$ cannot contribute as much to the RMSD as in the unbiased simulation. This means the RMSD with the largest range of orientations occurs at a lower value, also causing the histograms to peak at a lower value, and there is a cutoff on the highest RMSD expressed. The histograms from the historied simulations have a similar shape to the one from the unbiased simulation, but they are more compact like those from the memoryless simulations.

The profiles plotted in figure 2.12 provide more insight into the observed behavior in the historied bias simulations. The black curve represents the average density, $\langle \rho_{sim} \rangle_t$, at the end of the 100 ns simulation. Similar characteristics were prevalent in the running average as early as a few nanoseconds from the start of the simulation. $\langle \rho_{sim} \rangle_t$ tended to overfill in the center, relative to ρ_{exp}, and underfill at the ends of the profile. The reason for this trend is easily understood; of all conformations with $\mu^{(1)}_{sim}$ localized near $\mu^{(1)}_{exp}$, most contribute density at $\mu^{(1)}_{exp}$. This means that most of the weight of the density

profile is at the center. Another way to put it is that convolving a density that roughly look like ρ_{exp} (the aspect ratio will fluctuate as peptide orientation changes) with the function that describes the distribution of $\mu^{(1)}_{sim}$ (a roughly symmetric distribution centered on $\mu^{(1)}_{exp}$) will lead to a profile similar to that plotted in figure 2.12.

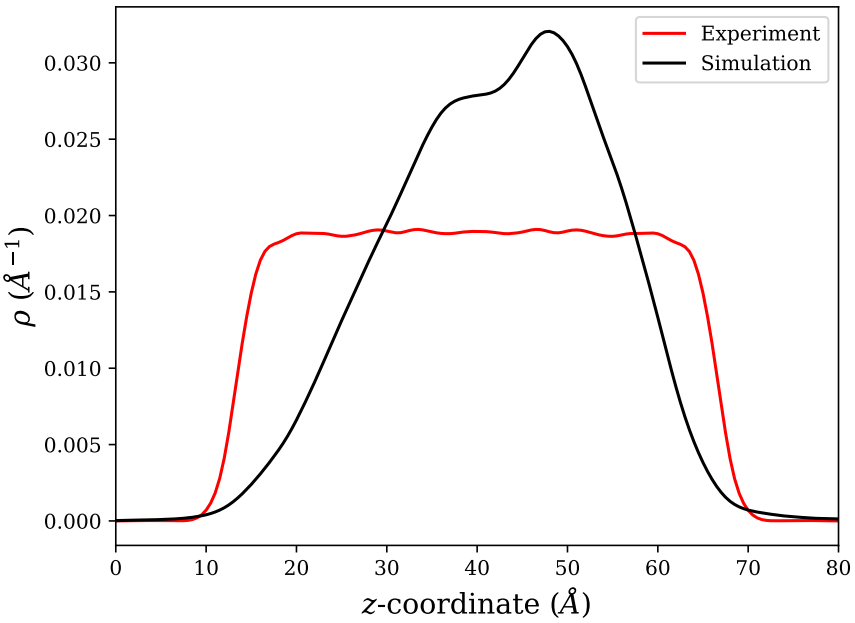

Figure 2.12: Comparison of target profile for pALA simulation (red) with the resulting time-averaged profile for the historied bias (black), $\lambda = 0.5\ k_BT \cdot nm$. Subtracting the target experimental profile from the average simulation profile is equal to the bias potential at the end of the simulation, up to a scaling factor.

$\langle \rho_{sim} \rangle_t$, calculated from simulation start to time step under consideration, contributes to the historied bias. With the profile shown, the resulting bias is weakly repulsive in the middle and weakly attractive at the sides. This is what generates the bimodality in the center position's histogram from figure 2.8. Also, because the attractive regions are narrower than the helical length, orientations off the z-axis are favored. The bias is weak enough that the central peak of ρ_{sim} persists despite its repulsive nature and the underfilled

sides persists despite their attractive nature. With larger λ, the bias would prefer conformations that fill the edges more and center less, leading to closer agreement with the target profile.

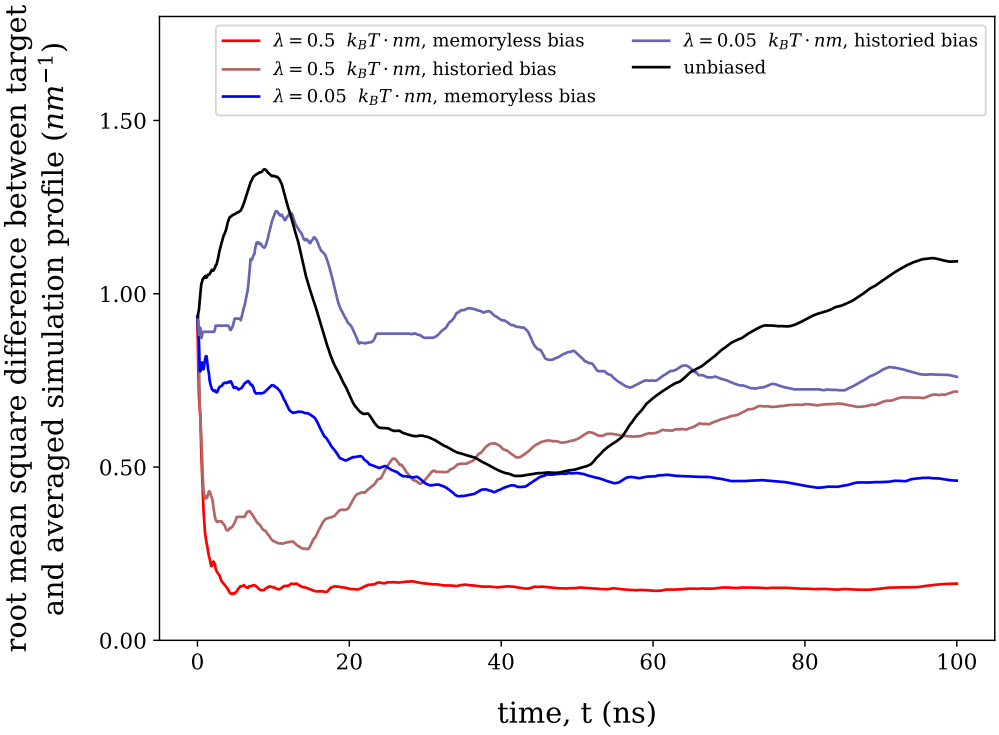

Figure 2.13: Root mean square difference between target profile (figure 2.7) and simulation profile averaged up to the current time step. The RMSD values for the memoryless biases (saturated colors) trend toward constant steady state values that are lower than the trend in the historied biases (unsaturated colors). The RMSD values in the hisoried biases trend toward similar asymptotic values. The RMSD in the unbiased simulation does not appear to have any convergent behavior.

Figure 2.13 shows, for each simulation, how $\langle \rho_{sim} \rangle_t$ instead of $\rho_{sim}(t)$ compares to ρ_{exp}. The time averaging produces curves which are much smoother than those observed in figure 2.9. The unbiased simulation shows an initial increase in RMSD, because the center position (figure 2.8) drifted away

from the target center position early on. Contributions in the middle of the trajectory matched better on average with ρ_{exp} until the time at which $\mu^{(1)}$ drifted to the edge the box for the remainder of the simulation. The RMSD increases steadily in this time range from a series of configurations that produce density in regions where ρ_{exp} is zero.

The RMSD for the strong memoryless bias reaches its lowest value within 5 ns and has some small fluctuations intially, smoothed out later by the large number of samples in the average. The RMSD for the weak memoryless bias has a similar behavior, taking longer to reach its asymptote and smooths out at a higher value. The strong historied bias behaves like the complementary memoryless bias at the start, but does not reach as low a value and the curve turns upward around $t = 15\ ns$. This bias does not hold the configuration space tightly to ρ_{exp} and over time seems to weaken its restraint as the peptide explores configurations that reduce the average RMSD. The RMSD of the weak historied bias also fails to drop as low as its same-strengthed memoryless bias early on. After a short time, the RMSD has an upturn similar to the unbiased simulation. Over longer times, the bias brings the average profile RMSD back down to a very similar value to that of the strong historied bias.

2.3.3 Summary

Both the memoryless bias and the historied bias produce a set of configurations with a resulting ρ_{sim} closer to ρ_{exp} than the density from the unbiased simulation, as measured by the RMSD. The memoryless bias restricts both the center position of the peptide and its orientation, resulting in low RMSD values throughout the simulation. The RMSD calculated using a time-averaged density also shows stronger agreement in the memoryless biases than in the historied biases of the same strength.

The trajectory in the historied bias explores a larger range for both center positions and orientations of the peptide, though the configurations that make up the mock profile did not vary these degrees of freedom. The historied bias provided an alternate set of configurations for constructing the target profile. Though this did not lead to an RMSD as low as for the memoryless bias, it peturbs the phase space less relative to the unbiased simulation.

2.4 Phosphatase And Tensin Homolog (PTEN)

2.4.1 Description

PTEN is a lipid phosphatase whose association with stBLMs *in vitro* has been thoroughly studied with NR[88] and, independently, with MD simulations.[92,93] CVO profiles derived from the NR experiment are shown in figure 2.14. Fortuitously, CVO profiles extracted from experiment and simulation agreed so well that there was little doubt that the MD simulations retraced the experimental results. While the biological significance[94–97] and structural details of PTEN[98] are beyond the scope here, the relevant parts of the structure include two folded domains (an N-terminal phosphatase domain and a Ca^{2+}-independent C2 domain) and a disordered C-terminal tail which accounts for about 13% of the entire molecular weight and controls membrane-accessibility of C2.[99,100] The sequence of the core and tail regions can be found in figure 2.15

An MD simulation was performed with a bias deriving from NR results obtained for a membrane composed of the phospholipids 1,2-dioleoyl-3-phosphatidylcholine and 1,2-dioleoyl-3-phosphatidylserine with cholesterol as a minor component (DOPC/DOPS/chol = 67:30:3). In the starting configuration, taken from a previous simulation,[93] the PTEN protein was moved ca. 35 $\AA$ away from the bilayer and solvated with 100 mM NaCl in TIP3P water (CHARMM 36, July 2017 forcefield) in a cubic box with dimensions (16.04, 16.04, 21.35) *nm*. A 5000 step energy minimization was performed, followed by a series of simulations with restraints on various groups within the system that were relaxed sequentially. The restraint relaxation protocol was supplied by the *CHARMM-GUI Membrane Builder* (`http://www.charmm-gui.org/?doc=input/membrane.bilayer`) and are shown in Table 2.1. The system was then simulated for an additional 1ns run to check to equilibration of the system before production.

Subsequently, all production runs were started under the bias of the profiles centers according to eq (2.42), because the condition for spatial disjointedness, eq (2.41), applied. They were then either subjected to steering with a potential based on a profile for the protein envelope at the membrane previously determined.[88] For 'unbiased' simulations, the value of λ in eq (2.37), was set to zero, but the protein was drawn to the membrane based on the profile center bias mentioned.

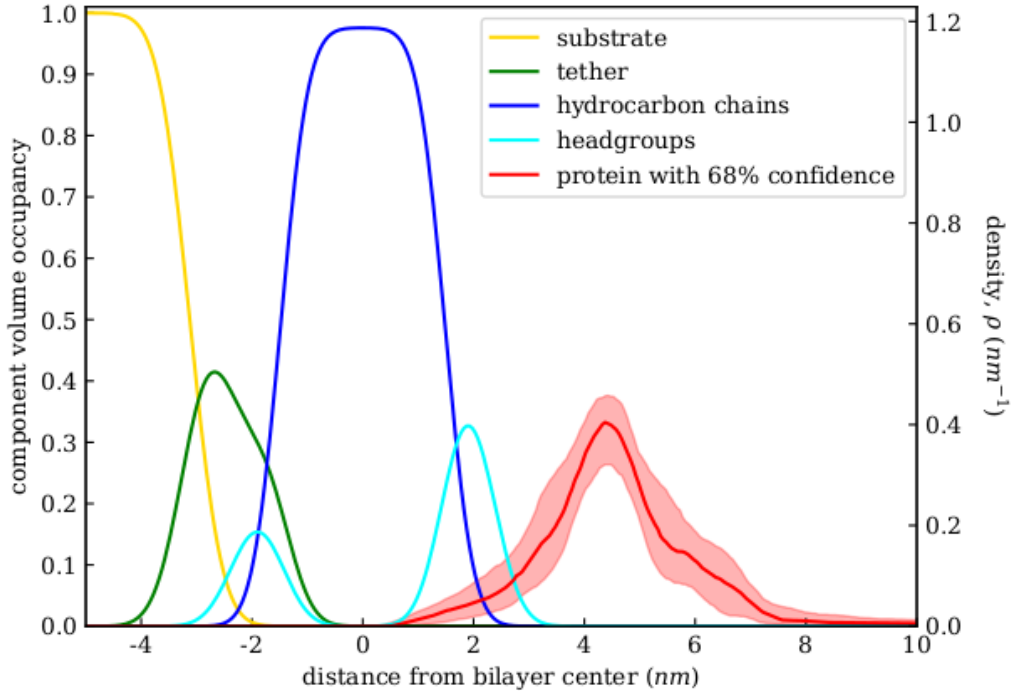

Figure 2.14: Component volume occupancy fit from NR experiment of PTEN on an stBLM (DOPC/DOPS/chol = 67:30:3).[88] Right axes shows the scaling of the CVO used in biasing, such that the area under protein profile is equal to one.

Step	Time	Protein		Lipids	
		Backbone	Sidechains	Position	Dihedral
	ps	$kJ/(mol \cdot nm^2)$	$kJ/(mol \cdot nm^2)$	$kJ/(mol \cdot nm^2)$	$kJ/(mol \cdot rad^2)$
1	25	4000	2000	1000	1000
2	25	2000	1000	1000	400
3	25	1000	500	400	200
4	100	500	200	200	200
5	100	200	50	40	100
6	100	50	0	0	0

Table 2.1: Restraints used in equilibration of the PTEN system, all forms are harmonic and can be found in the GROMACS documentation.

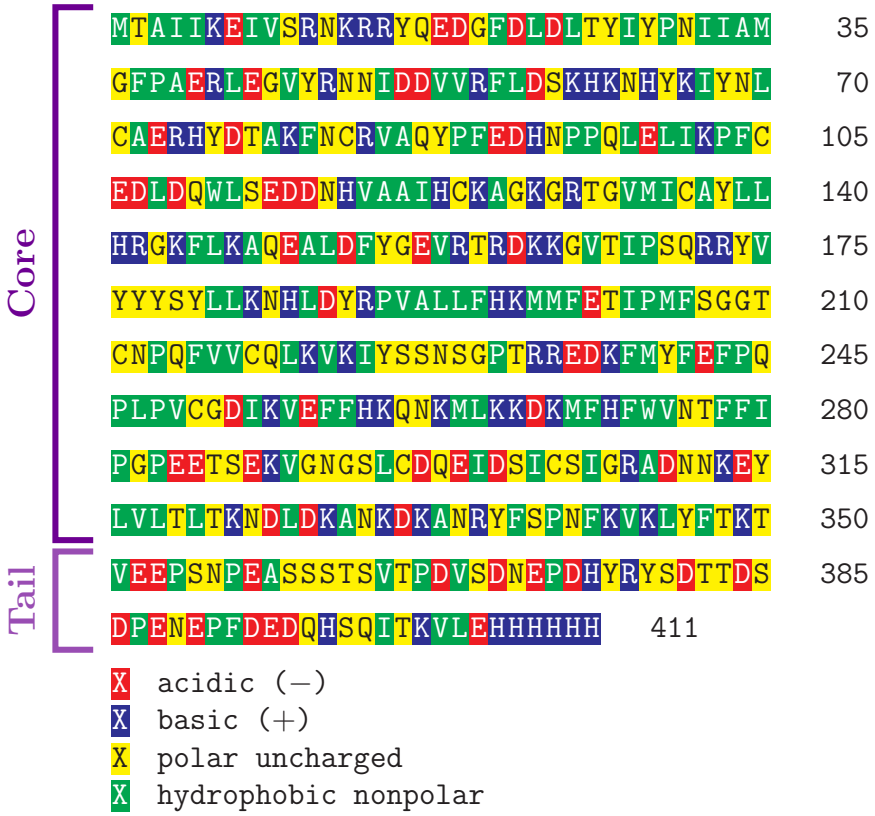

Figure 2.15: PTEN sequence. Amino acids are colored by their charge. Rigid core domain (residues 1 to 350) and flexible tail domain (residues 351 to 411) are labeled at left of sequence.

2.4.2 Results

Figure 2.16 shows the center position of the profiles over time. All biased simulations were nearer to that of the target profile's center, $\mu^{(1)}_{exp} = 4.55\ nm$, than the unbiased simulation. The historied bias simulations were nearer to the membrane than the memoryless bias simulations. This may be in part to the difference in the density contributions from the tail region (see figure 2.15 for region definition). The tail region and the consequences of its density contribution will be described later in this section, along with a depiction of this density in figure 2.21.

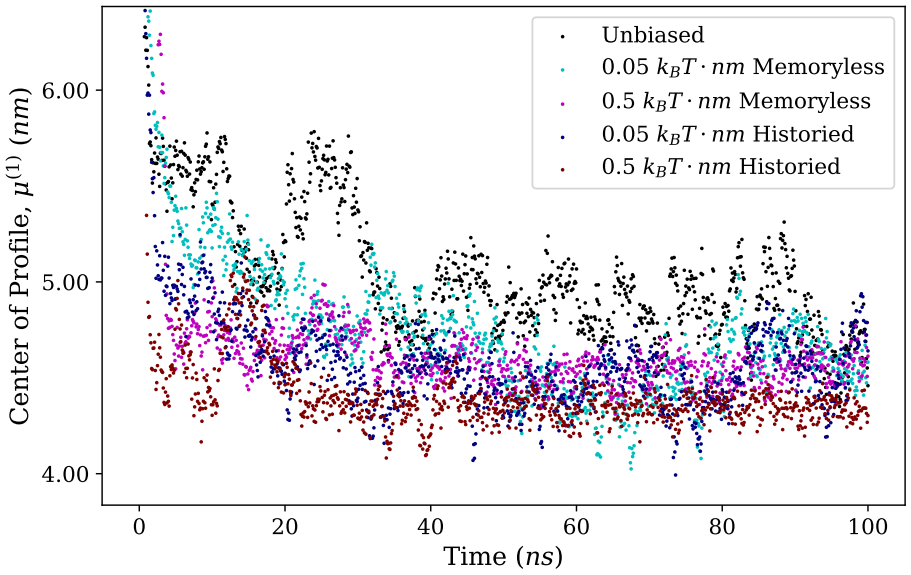

Figure 2.16: $\mu^{(1)}_{sim}$ for the simulations of PTEN, measured relative to the bilayer center. $\mu^{(1)}_{exp} \approx 4.55\ nm$. All biased simulations have center positions nearer to the experimental value than the unbiased simulation.

Figure 2.17 shows the RMSD between the simulation and target densities at each time step. From around 40 ns onward, the weaker bias does not differ in its RMSD fluctuations from the unbiased simulation. The stronger bias, however, shows unexpected behavior. The fluctuations for both the historied and the memoryless bias are smaller with the memoryless bias showing the

lowest RMSD of all five simulations. But, the memoryless bias transitions between 20 ns and 25 ns from having a low RMSD to steadily having the highest RMSD of all the simulations without much fluctuation in the RMSD. The structural details that lead to this jump in RMSD will be examined later in the section.

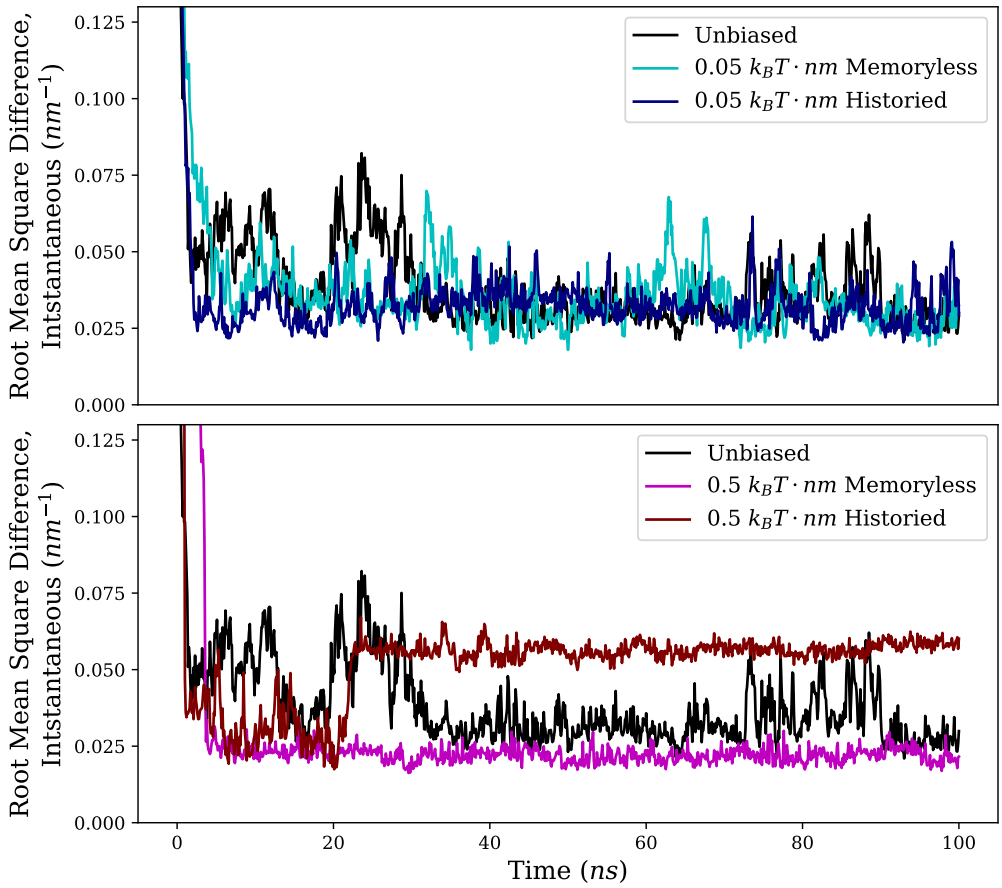

Figure 2.17: RMSD between ρ_{exp} and ρ_{sim} for each output frame of the PTEN simulations. The strong historied bias shows unique behavior, transitioning between a state of low RMSD ($\approx 0.025\ nm^{-1}$) and a state of high RMSD ($\approx 0.060\ nm^{-1}$) between $t = 20\ ns$ and 25 ns.

Figure 2.18 shows the RMSD values for each simulation using the running average, $\langle \rho_{sim} \rangle_t$, the trends of which behave as might be expected given

the instantaneous RMSD. Each RMSD drops intially as spatial overlap is increased between the simulation profiles and the experiment. Also the trajectories become more smooth as the number of samples increase. For all but the strong historied bias, the curves trend toward lower RMSD values over time. The strong historied bias's curve turns upward after $20ns$, related to the jump transition in figure 2.17 for this simulation, eventually reaching the highest sustained averaged RMSD value of all the simulations.

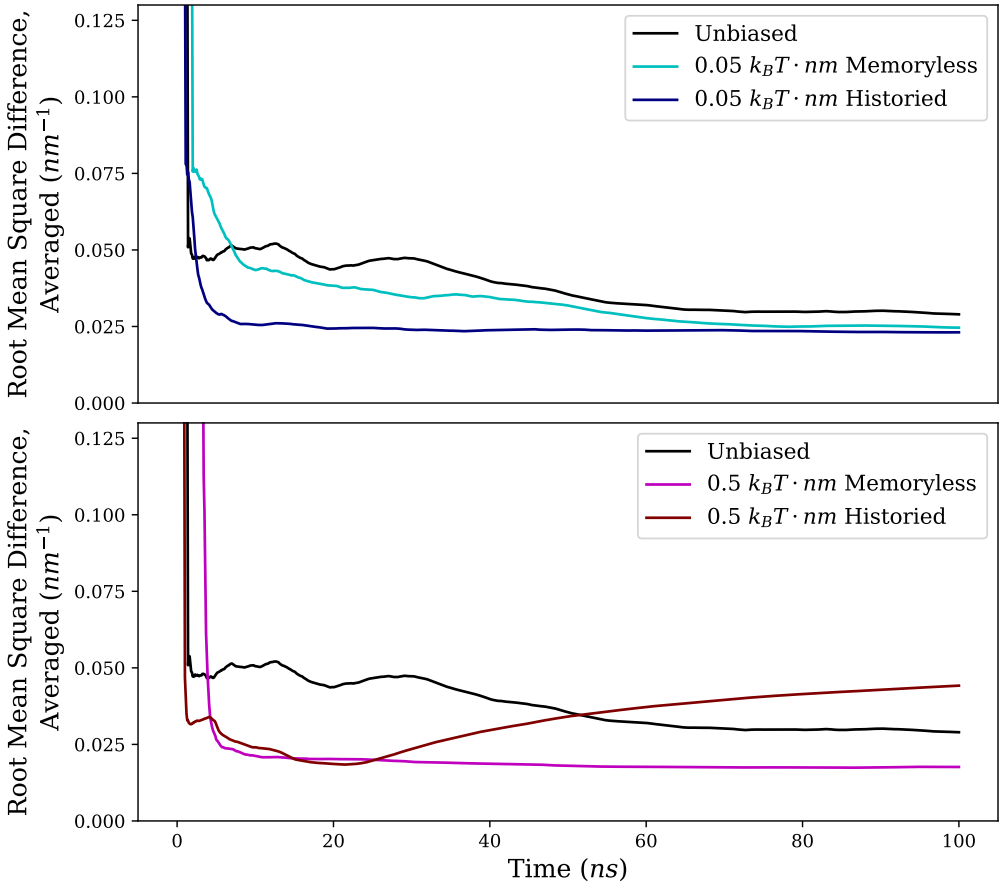

Figure 2.18: RMSD between ρ_{exp} and $\langle \rho_{sim} \rangle_t$ for PTEN simulations. All simulations except the strong historied bias have RMSD values that decrease over time.

Shown in figure 2.19 are the RMSD values between the core domain atoms

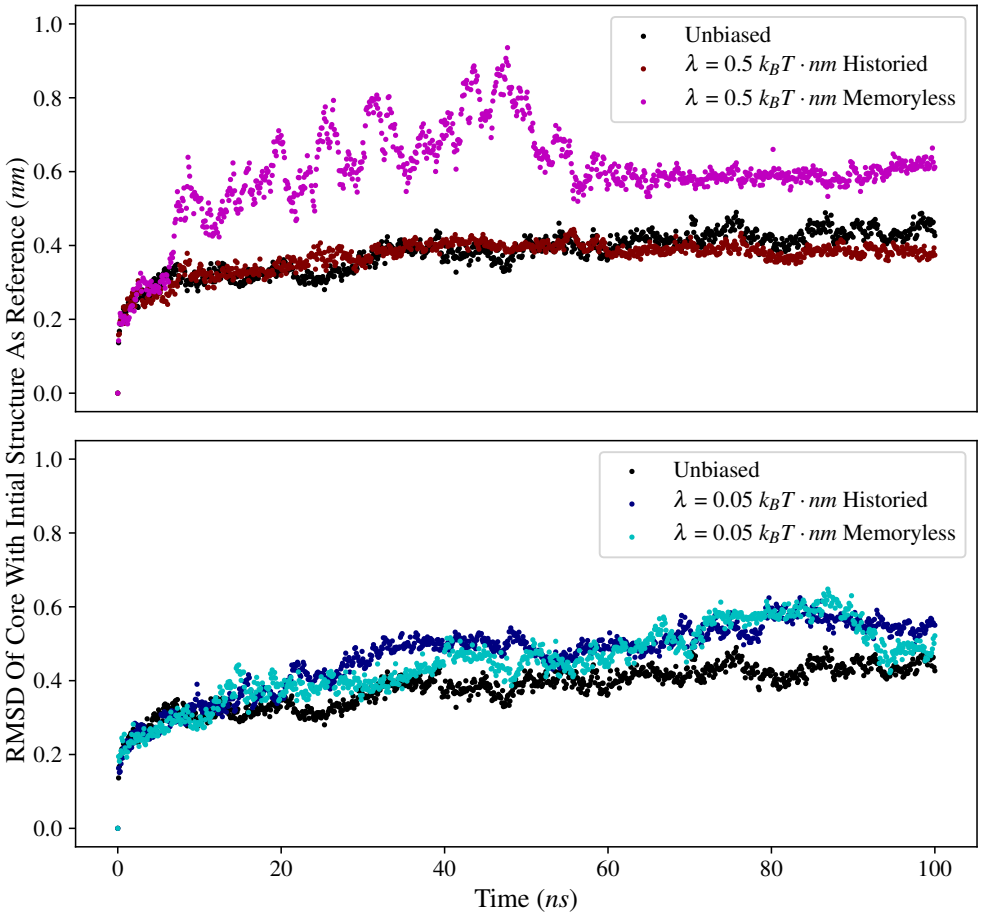

Figure 2.19: RMSD of atoms in the core of PTEN (residues 1–350) from the starting structure at each time. The atoms belonging to the core region of PTEN in the strong memoryless bias deviate more from their initial positions than those in any other simulation. The strong historied bias has values most similar to the unbiased simulation.

in the intial structure and their position at each output frame of the simulations. This describes how the internal structure changes throughout time, using the initial structure for reference. The sustained difference in RMSD between the strong memoryless bias and unbiased simulation toward the end

of the run is greater than the difference seen between either weak bias and the unbiased simulation. (0.6 nm, 0.5 nm, and 0.4 nm). The RMSD of the strong historied bias appears to plateau, instead of continuing to increase like the others, and is consistently lower than the unbiased simulation between 60-100 ns. Given the rigidity of the folded core domain, the RMSD behavior suggests that the structure in the strong memoryless bias is more distorted than the other simulations and that the core structure of the strong historied bias may be slightly more stablized than the others.

To evaluate how well different biasing strategies steer simlations most efficiently towared the desired outcome, the most informative metric is a direct comparison of the density profiles. This illustrates not only the amount to which there is agreement, but also where in the protein discrepancies occur. Figure 2.20 shows, for each run, the density from the final 75 ns of the trajectory. The shading of the simulation densities shows the distribution of densities observed in this time range. Opacity is used to convey fluctuations about the median observed density, the transparency increases with distance from the median.

In the unbiased simulation, the peak of the protein profile does not quite fill the experimental density but falls within the confidence interval of the fit for nearly all densities observed. At higher values of z, around 18 nm, there is an excess of density relative to the experimental profile, rising above the confidence interval of the fit for most densities observed in simulation. The higher z values that are over-represented in the unbiased simulation have lower densities in the biased simulations. The memoryless simulations retain some of the density here, but shift the weight toward the peak as observed in the experimental density.

Figure 2.21 provides a more detailed structural explanation for the variation in agreement between the various simulations and the experiment. The five simulation densities are plotted in the same way as in figure 2.20 except that the contributions from the rigid core domain (darker) and flexible tail domain (lighter) are plotted separately. Only in the unbiased simulation is the majority of the tail density located at higher z-values than the core, away from the membrane. Some of the density around $z = 4.5$ nm is attributed to where the the tail joins to the core. But excluding this small amount, one still concludes that the tail was more often closer to the membrane in the biased simulations than in the unbiased.

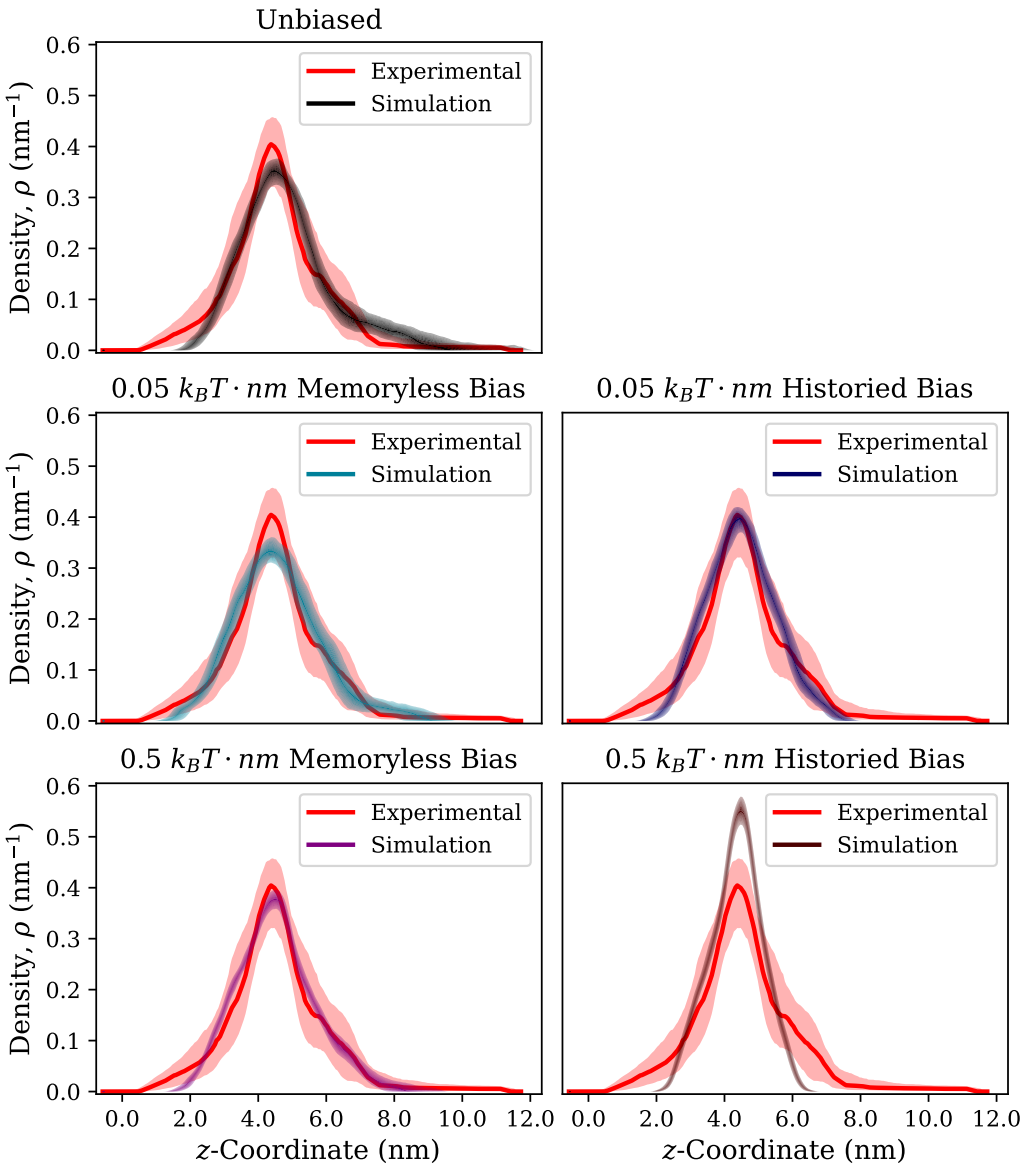

Figure 2.20: Protein density profiles observed in PTEN simulations are compared with NR derived protein distribution on bilayer (67:30:3 DOPC:DOPS:cholesterol.)[88] that was used for biasing. For the experimental profile, the median of the fit to NR data is plotted (solid line, red) along with the 68% confidence interval for that fit (shaded, red). Simulation densities plotted occur in the last 75 ns of thier 100 ns runs. In the simulation profiles, shading conveys the distribution of densities observed at a point z. Full opacity is used at the median, and full transparency is used at the tails of the observed density distribution (a cutoff is made beyond the 68th percentile).

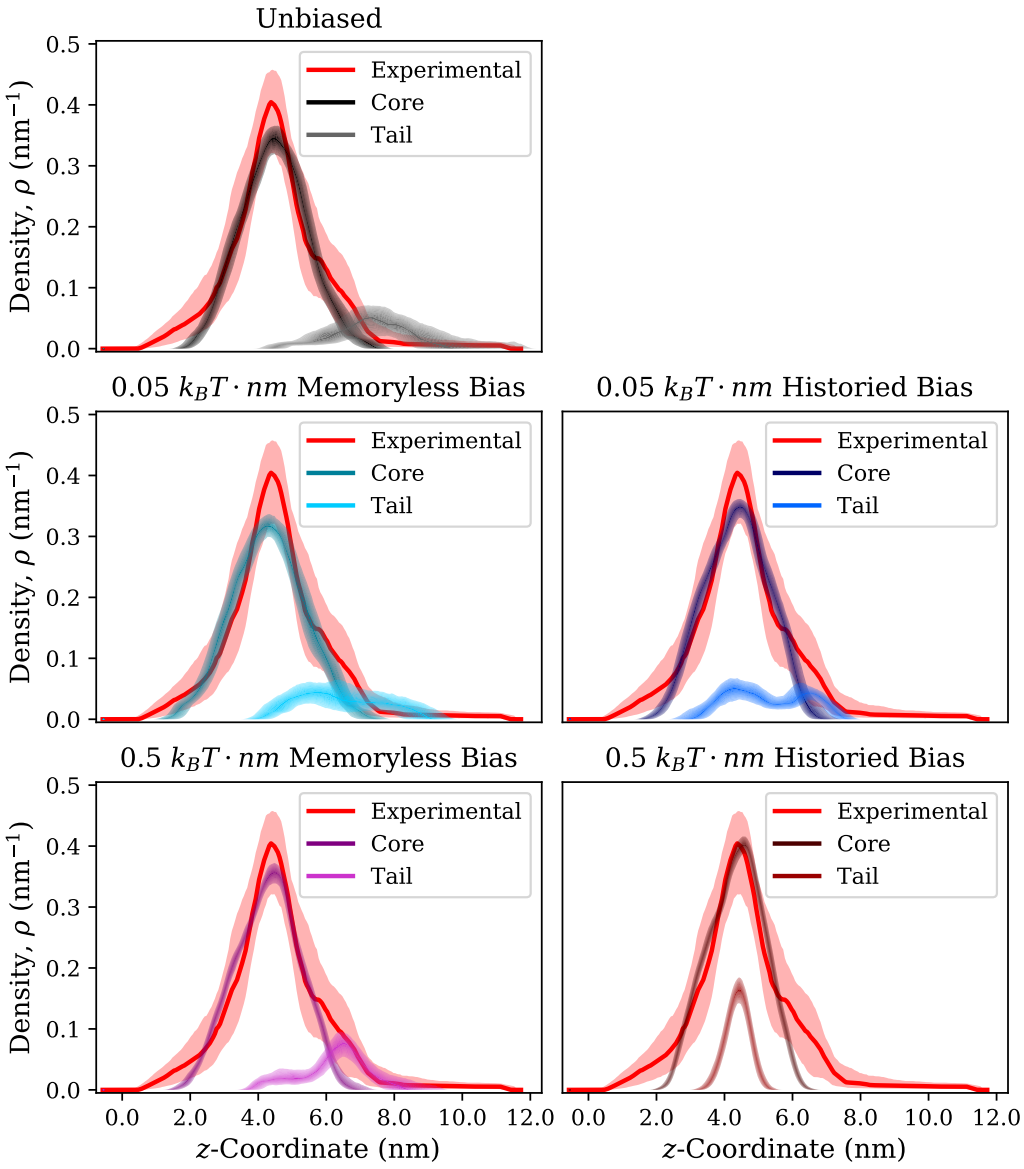

Figure 2.21: Density profiles from PTEN simulations (25-100ns) and experiment with simulation profile divided into core region (residues 1–350) and tail region (residues 351–411). For the experimental profile, the median of the fit to NR data is plotted (solid line, red) along with the 68% confidence interval for that fit (shaded, red). In the simulation profiles, shading conveys the distribution of densities observed at a point z. Full opacity is used at the median, and full transparency is used at the tails of the observed density distribution (a cutoff is made beyond the 68th percentile).

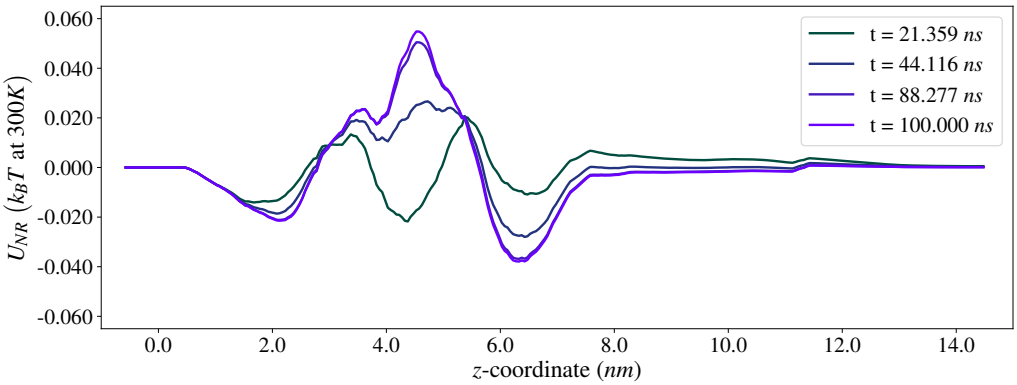

Figure 2.22: Bias as calculated from the average density reported in the checkpoint files (used for restarting simulations) of the strong historied simulation. Checkpoint files contain the average of all densities sampled, containing 1000 times as many samples as the outputted trajectory. Difference of the average density and experimental density scaled by $\lambda = 0.5\ k_BT \cdot nm$ at four times, showing how the bias is approaching a seemingly steady-state value. The bias at 66.489 ns is omitted due to its indistinguishability from that at $t = 88.277\ ns$.

The excess simulation density observed around 4.5 nm in the strong historied bias contributes to a peak in the bias at this location, seen in figure 2.22. This penalizes the extra density and should repel atoms to reduce the excess density. The flexibility of the tail would suggest that it could easily be repelled away from the bilayer by this peak. However, the tail remains in this region even as the peak of the bias grows throughout the course of the simulation. A naive explanation would be that the tail is repelled to smaller z and becomes trapped between the peak in the bias and the membrane. In reality, the bias is not so large ($U_{NR} < 0.06\ k_BT$) that the tail would be trapped below the peak. This conclusion is further supported by the fact that the tail density is centered on this peak. The observation of this tail behavior appears to be of an origin other than the bias, perhaps a complicated interaction between the tail and membrane, and its source beyond the scope of this work.

Figure 2.23 shows illustrative examples, from two perspectives, of the historied simulations and how their difference in structure contributes differently to the tail densities. The core is colored red and the tail is colored blue.

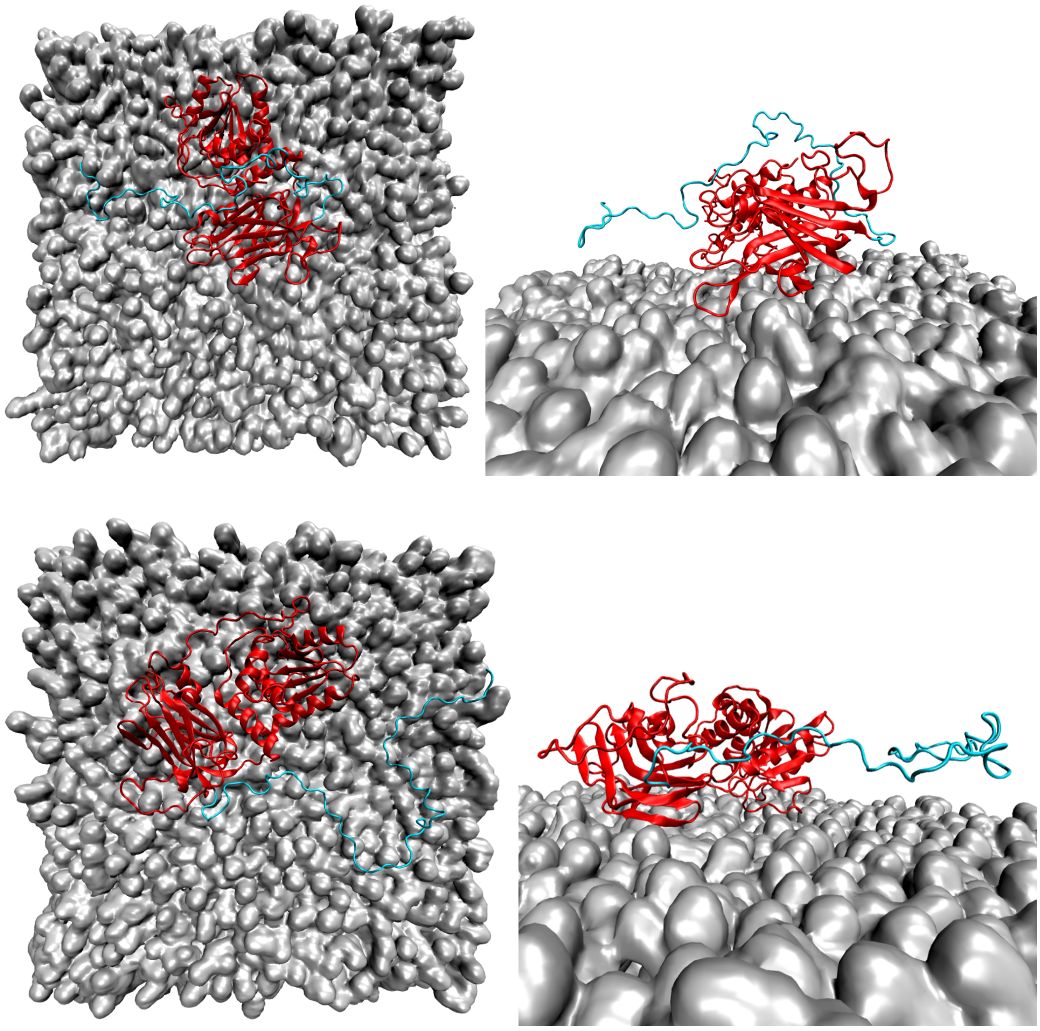

Figure 2.23: Two orientations for a single snapshot from each simulation with historied bias. Top: Weaker bias at time $t = 90.3\ ns$, showing an example of the of the tail (cyan) orienting over the top of the protein core (red). Bottom: Stronger bias at time $t = 67.8\ ns$, showing the observed tendency of the tail to stay near to the membrane after 20 ns.

After 25 ns in the simulation with stronger bias, the tail is primarily observed to be extended along the membrane with small variation in its height off the

membrane. This leads to a single, narrow peak in the tail density. In the weak bias, the tail explores the membrane on the opposite side of the core by wrapping itself over the top. This makes a bimodal contribution rather than the single narrow peak observed with the stronger bias. It should be noted that these snapshots should be taken as an intuitive guide, the density is an average that also involves the dynamical nature of the flexible tail. Figure 2.23 does, however, capture an important difference in the two simulations.

The memoryless bias simulations have a simpler explanation for their dynamics than the historied bias simulations. They can be explained by the idea that the memoryless bias works to suppress fluctuations from ρ_{exp}, this action being more apparent in the stronger bias than the weaker. One effect of the bias on the protein is to slightly tilt the core in the case of the stronger bias. In the unbiased simulations, the core density matches the profile well on its own but skews to higher z values when tail density is added. Tilting the core counters the skew of the density toward larger z values. Another effect of the bias is to somewhat restrict the tail to roughly point in one direction and maintain its length of extension, enforcing agreement with the density at z values above the peak. There are fluctuations, of course, but the dynamics are far less varied than that of the historied bias.

2.4.3 Summary

As shown in figures 2.17 and 2.18, all biased simulations resulted in better agreement between ρ_{exp} and ρ_{sim} or $\langle\rho_{sim}\rangle_t$ (based on RMSD), with the exception of the strong historied simulation. The best agreement was observed with the strong memoryless bias and this agreement is well represented by the closeness with which the simulation density follows the experimental curve in figure 2.20. Unfortunately, the shading of the data indicates that this simulation fluctuated within a smaller range of densities than the unbiased simulation. In addition, the RMSD between the positon of core atoms in the starting structure and throughout the simulation (figure 2.19) has distinctly larger values for the strong memoryless bias than for the other simulations. This indicates that the bias may apply significant stress to the internal structure of the core domain in this case.

The next strongest agreement, throughout the entire simulation is for the weak historied bias. The weak memoryless bias reaches similar agreement near the end of the simulation based on RMSD of $\langle\rho_{sim}\rangle_t$. The larger fluctuations in the RMSD of the instantaneous ρ_{sim} and broader distribution of

densities sampled (figures 2.20 and 2.21), especially in the tail, suggest that this simulation may be sampling a slightly larger range of conformations that over time contribute a density that averages to a similar level of compatibility. It is difficult to argue which simulation captures more useful information, both are compatible and should perhaps be considered simultaneously.

The two simulations using weak biases also represent slightly different structural features. In the historied bias, parts of the tail associate with the membrane unlike in the memoryless bias. If there is an interaction which associates the tail with the membrane, this could lead to the smaller fluctuations in RMSD of ρ_{sim} considering that the tail wraps over the core (figure 2.23) and would help to anchor its orientation.

The observations in the simulation using strong historied bias provide more evidence for membrane-tail association. Even though the simulation density disagrees with ρ_{exp} and there is an energetic penalty for this disagreement, the tail is associated with a region near the membrane that has the least favorability in the potential. It seems that instrinsic forces, independent of the bias, keep the tail in that position. The core density, however, closely matches the experimental density around the peak (figure 2.20). The source of this matching is how the core is tilted differently from other simulations. If there is an interaction of the tail with the membrane that applies a tensile force, this could assist in tilting of the core. This assistance from the tail would alleviate stresses on the core from the bias, which would explain how the atomic RMSD (figure 2.19) shows the best agreement with the unbiased simulation. While providing the least agreement with ρ_{exp}, the simulation using strong historied bias provides interesting behavior worth further investigations.

2.5 Discussion (Items Of Interest And Future Directions)

As demonstrated in the previous sections, the bias constructed for NR density profiles improves the agreement between the data and the simulation. However, it is still unanswered whether this method is the most effective way to bias the simulation using experimental data. A few points are addressed in this section with regard to the potential and possible alternatives. Examined are the implications of using a time-average in the bias and the use of a

single value for λ at all points in the density.

2.5.1 Effect Of Time Averaging On Effectiveness Of λ

As the simulation progresses, more samples contribute to the average used in the historied bias. This reduces the significance of any single configuration within the bias. Additionally, if sampling is frequent enough, a set of sequential configurations contribute similar densities. The average density for biasing comes from eq (2.38)

$$\langle \rho_{sim} \rangle_t = \frac{1}{M} \sum_{i=1}^{M} \rho_{sim}\big(t_0 + i\tau\big) \tag{2.38}$$

If N is the typical number of samples that can be treated as having similar densities, the average over a subinterval with N samples,

$$\overline{\rho_{sim}}(t) = \frac{1}{N} \sum_{j=0}^{N-1} \rho_{sim}(t + j\tau) \ , \tag{2.45}$$

is approximately equal to any density in the subinterval:

$$\overline{\rho_{sim}}(t) \approx \rho_{sim}(t) \ . \tag{2.46}$$

Rearranging the sum in eq (2.38) and using eqs (2.45) and (2.46) to rewrite the average with a different sampling rate,

$$\begin{aligned} \langle \rho_{sim} \rangle_t &= \frac{1}{M} \sum_{i=1}^{M/N} \sum_{j=0}^{N-1} \rho_{sim}\big(t_0 + (Ni + j)\tau\big) \\ &= \frac{1}{M} \sum_{i=1}^{M/N} N \ \overline{\rho_{sim}}\big(t_0 + Ni\tau\big) \\ &\approx \frac{1}{M/N} \sum_{i=1}^{M/N} \rho_{sim}\big(t_0 + i(N\tau)\big) \ , \end{aligned} \tag{2.47}$$

shows that the average is almost unchanged for a sampling time which is longer by a factor of N. This means that the average effectively contains

only some maximum number of samples, not changing in response to a faster sampling rate.

The number of samples in the average affects the response of the bias to any deviation from ρ_{sim}. Specifically, an analysis similar to that used in section 2.1.3 shows how the number of samples changes the strength of the bias. The same potential for a particle in the one-dimensional well defined in eq (2.21) is used with nearly the same sampling procedure outlined in appendix B.3.1.

$$\beta U_0(q) = 25(q - 0.25)^4 - q\cos(q) + \frac{sin(20q)}{q^2 + 0.5} \tag{2.21}$$

Instead of imposing the condition that the average q among replicas be the target value, which corresponds to an infinite harmonic restraint on the average, a linear bias will be applied to the average q. This bias will be added to the potential used in the metropolis sampling algorithm and proposal distributions for each replica will be independent uniform distributions centered on the current q value.

$$U(q_i) = U_0(q_i) + \lambda\left(-Q + \frac{1}{N}\sum_{j=1}^{N} q_j\right) \tag{2.48}$$

Figure 2.24 shows the effective bias on the simulation produced for a few values of λ and number of replicas. For the four simulations shown, only two distinct biases are seen. In each case, the result of the bias is equivalent to applying linear bias with a single replica but not necessarily with the same strength. The strength of the effective linear bias depends on the ratio of λ to number of replicas.

$$U_{eff}(q) = U_0(q) + \lambda_{eff} q \tag{2.49}$$

$$\lambda_{eff} = \frac{\lambda}{N} \tag{2.49'}$$

As the number of replicas increases, the effective linear bias decreases on the entire distribution sampled. The same behavior is seen clearly in figure 2.13 with the RMSD for $\langle\rho_{sim}\rangle_t$ in the pALA simulation with historied bias, $\lambda = 0.5\, k_B T \cdot nm$. As time goes on, the average density deviates further away from ρ_{exp} as the number of samples contributing to the bias increases.

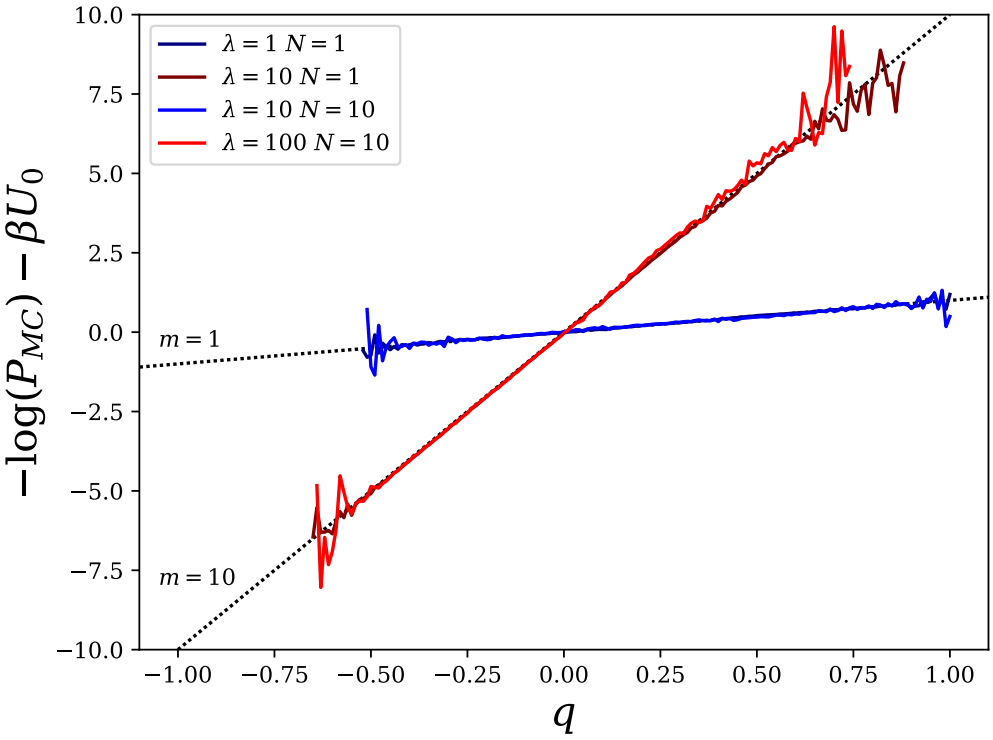

Figure 2.24: Effective bias added to Monte Carlo simulation. For each set of parameters, the unbiased potential is subtracted from the effective potential as determined by the probability density produced from sampling. Dotted lines with the indicated slope are included for reference.

However, if the sampling rate is faster than the response of the observable, eq (2.47) indicates that the number of samples is not tied to the algorithm for averaging but is dependent on the evolution of the system. This means that the time scale for relaxation of the RMSD should be the same for sampling at a rate other than $\tau = 0.1$ ps. Conversely, the time scale of the RMSD decay gives insight into the time between distinct contributions to the density for the system under study.

There are a couple ways in which this effect can be managed. The simplest is to scale λ with the progression of time in the simulation. Another option

is to use an *exponential moving average*:

$$\langle\rho\rangle_t\Big|_{t=t_0+i\tau} = \alpha\cdot\rho(t_0+i\tau) + (1-\alpha)\cdot\langle\rho\rangle_t\Big|_{t=t_0+(i-1)\tau} \ , \tag{2.50}$$

where $\alpha \in [0,1]$ sets the exponential decay of the contribution from old samples. Rather than scaling the strength, a moving average keeps a fixed number of effective samples in the average. The first method is more similar to the method of restrained ensemble and the second requires less modification to the existing code.

Both methods require some knowledge of the effective rate of sampling. Scaling of λ should be proportional to the effective number of samples. α should set a decay time dependent on the number of effective samples to be retained in the average. The effective rate of sampling is difficult to predict and varies between systems. However, the RMSD between the density from simulation and experiment will respond to the scaling of λ and could be used to indicate a rescaling or size of α.

2.5.2 Spatial Dependence Of λ

If the bias is thought of as a Lagrange multiplier on the system Hamiltonian, it formally will have a dependence on z. This is also a natural conclusion in the mode of restrained ensemble since the density profile is a collection of points and the solution to the variational problem may also depend on which point in the density is being considered.

Additionally, ρ_{exp} is the result of a fit to data and inherently has a probabilistic interpretation. The density has confidence intervals at each point along z and each point can be thought of as having a probability distribution associated with it. This further emphasizes the z-dependence of λ, as the difference in prior knowledge about the density at each point informs how much the unperturbed simulation should be biased at that point in the density.

One way to include information about the fit in the bias is to scale λ according to the certainty of measurement at each point. If the certainty of density point is related to some range of probable values, $\Delta\rho$, then a simple implementation of a scaling function, $\Lambda(z)$, with values between zero and one

$$\Lambda(z) = \begin{cases} 0 & \text{if } \Delta\rho \geq \rho \\ 1 - \frac{\Delta\rho}{\rho} & \text{if } \Delta\rho < \rho \end{cases} \tag{2.51}$$

could multiply the constant λ to give spatial variation to the bias.

If a range of probable values for the density is larger than the value of the density, $\alpha = 0$ corresponds to having no reliable information with which to bias. It would also be reasonable to choose twice the range or some other multiple for the cutoff. It may also be desirable to have Λ be closer to one for some range about $\Delta\rho = 0$. In this case, a quadratic or other functional form that peaks around zero is a choice. Alternatively, one could even make $\alpha(z)$ adaptive based on some relationship between $\Delta\rho(z)$ and the distance, $|\rho_{sim}(z,t) - \rho_{exp}(z)|$ as the simulation progresses.

2.5.3 Other Functional Forms Of Bias And An Extension To The Restrained Ensemble

Other functional forms of biasing potentials are suitable for the task of generating agreement between experiment and simulation. The linear bias is the simplest and derives from the generalized solution to the variational problem in section 2.1, addressing how to minimally affect the unbiased probability distribution for the system. Also discussed in that section was the use of a quadratic potential as a bias. After cleverly modifying the application of bias using the technique of restrained ensemble, the quadratic potential also generated the minimally peturbed distribution.

A quadratic bias, or other functions of the form $f(\rho_{sim} - \rho_{exp})$, are options for incorporating NR data into simulation. With a quadratic bias, the restrained ensemble procedure can be implemented with replica simulations.[89] If replica simulations are too computationally expensive, a procedure akin to restrained ensemble could be attempted through the use of the time-averaged density, $\langle\rho\rangle_t$,

$$U_{NR} = \frac{k}{2}\left(\langle\rho_{sim}\rangle_t - \rho_{exp}\right)^2 \tag{2.52}$$

with the number of samples in the average replacing the replicas. When using samples from a time average, the rate of sampling as discussed in section 2.5.1 must be considered to count the 'replicas'. Finally there is a question of whether or not the amount of computation is reduced by using a time average. The length of a simulation necessary to produce enough distinct samples for the time average may amount to the same load as running replicas in parallel.

Appendix A

Code Base

A.1 GROMACS

The version of GROMACS used as of the publication of this thesis is 5.1.1.

A.1.1 How To Use The Modified GROMACS

The source code can be found for the memoryless version of the bias at:
https://github.com/bradtreece/gmx_instantaneous

For the historied version used throughout this thesis, use the branch **version_1**:
https://github.com/bradtreece/gmx_time_average

After downloading the repository, the code can be compiled by navigating to the build directory and remove all files if not already empty. The three machines where this code was tested and verified stable, the following program versions of compilation software were used.

machine	gcc	cmake	mpi	programming environment module (titan only)
local workstation (ubuntu)	7.4.0	3.10.2	openmpi/3.1.1	N/A
titan (ornl)	5.3.0	3.6.0	not recorded	PrgEnv-gnu
bridges (psc)	5.3.0	3.5.2	gcc_openmpi	N/A

The following cmake command is used to construct the cmake files and make

the executables from the build directory:

```
$ cmake .. -DGMX_BUILD_OWN_FFTW=ON
    ↪ -DREGRESSIONTEST_DOWNLOAD=ON -DGMX_MPI=ON
    ↪ -DGMX_GPU=OFF -DBUILD_SHARED_LIBS=OFF
$ make -j 8
```

After compilation of the executables, much of GROMACS operates as normal. The only modification to normal usage is with the routine **grompp**, when preparing '.tpr' files for usage with **mdrun**. The configuration ('.mdp') file supplied will contain the entry **userstr1**, which is used to refer to a user constructed file (described below) containing the information about the neutron potential. The value of **userstr1** can be set to either the total or relative path to the file. Do **not** include quotation marks around the file name.

The contents of the file containing the neutron data should be in the following format:

```
u       0.125   50.0
n       1       6649
p       10.09   22.14   0.05
d       0.0     0.0     0.0     0.0     0.0011  ...
```

The first character of each line (*u, n, p, d*) are flags for the preprocessor to recognize what is contained on that line. In the order shown, they are the scaling factors of the potential (λ, k), the range of indices for atoms to be forced (as found in the gro file), the parameters for the array describing the z-axis (z_{min}, z_{max}, z_{step}), and the neutron density array. All the values shown are tab delimited, but any white space should function properly. These are properly scaled to the units of GROMACS (nm, kJ/mol, etc.).

After **grompp** is run, the neutron data is loaded as input into the '.tpr' file and can be verified using the GROMACS program **dump**. Usage of **mdrun** is unchanged and if the simulation is to be extended, using the **convert-tpr** program supplied by these modifications will transfer the neutron data to the new '.tpr' file.

A.1.2 List Of Variables

Variable	Source	Use
input_rec	GROMACS	Stores Information From Starting Configuration
x	GROMACS	XYZ Position Vector For Atoms
v	GROMACS	XYZ Velocity Vector For Atoms
f	GROMACS	XYZ Force Vector For Atoms
userstr1	Modification	MDP Option Indicating The File Containing Input Data
exp_dens	Modification	Array Containing Experimental Density
sim_dens	Modification	Array Containing Simulation Density (Used In Bias)
ens_dens	Modification	Array Containing Simulation Density (Tracks Time Average)
pot_indices	Modification	Bounds On Global Indices For Atoms To Be Forced
pot_params	Modification	Start, Stop, And Step For Density Array
pot_scale	Modification	Potential Scaling Factors
exp_mean	Modification	First Moment Of Experimental Profile
second_moment	Modification	Second Moment Of Experimental Profile
sim_mean	Modification	First Moment Of Simulation Profile
num_of_states	Modification	Number Of States In Simulation Average
z_bbox	Modification	Local Instance Of z-Axis Periodic Box Size

A.1.3 Files

List of Modified Files:

- source_dir/src/gromacs/fileio/tpxio.c
- source_dir/src/gromacs/gmxpreprocess/readir.c
- source_dir/src/gromacs/mdlib/sim_util.cpp
- source_dir/src/gromacs/mdlib/stat.cpp
- source_dir/src/gromacs/mdlib/broadcaststructs.cpp
- source_dir/src/gromacs/mdlib/md_support.cpp
- source_dir/src/gromacs/gmxlib/txtdump.c
- source_dir/src/gromacs/gmxlib/checkpoint.cpp
- source_dir/src/gromacs/gmxlib/typedefs.c

- source_dir/src/gromacs/legacyheaders/md_support.h
- source_dir/src/gromacs/legacyheaders/types/state.h
- source_dir/src/gromacs/legacyheaders/types/inputrec.h
- source_dir/src/gromacs/legacyheaders/sim_util.h
- source_dir/src/gromacs/domdec/domdec.cpp
- source_dir/src/programs/mdrun/md.cpp

A.1.3.1 tpxio.c

This file is involved in the writing of the run submission files with extension tpr (previously tpx). This is necessary for passing values (like experimental density) into the simulation. Changes were made within **do_inputrec** around line 1503.

```
1500     gmx_fio_do_real(fio, ir->userreal3);
1501     gmx_fio_do_real(fio, ir->userreal4);
1502
1503     gmx_fio_ndo_real(fio, ir->pot_indices, 15); //Brad
1504     gmx_fio_ndo_real(fio, ir->pot_params,   3); //Brad
1505     gmx_fio_ndo_real(fio, ir->pot_scale,    2); //Brad
1506     gmx_fio_ndo_real(fio, ir->exp_dens, 10000); //Brad
1507     gmx_fio_ndo_real(fio, ir->exp_mean,     1); //Brad
1508     gmx_fio_do_double(fio, ir->z_bbox);         //Brad
1509     gmx_fio_do_double(fio, ir->second_moment);  //Brad
1510
1511     /* AdResS stuff */
1512     if (file_version >= 77)
1513     {
```

A.1.3.2 readir.c

This file includes the reading of the configuration file (mdp) in order to construct the input record (input_rec or ir). The main body of the modification reads the input file containing the information pertaining to constructing the potential and processes it into the variables in the input record. The modified function is **get_ir**, around line 2296.

```
2293      CCTYPE ("User defined thingies");
2294      STYPE ("user1-grps",  is->user1,          NULL);
2295      STYPE ("user2-grps",  is->user2,          NULL);
2296      STYPE ("userstr1",    is->userstr1,       NULL); //
              Brad
2297      ITYPE ("userint1",    ir->userint1,   0);
2298      ITYPE ("userint2",    ir->userint2,   0);
2299      ITYPE ("userint3",    ir->userint3,   0);
2300      ITYPE ("userint4",    ir->userint4,   0);
2301      RTYPE ("userreal1",   ir->userreal1,  0);
2302      RTYPE ("userreal2",   ir->userreal2,  0);
2303      RTYPE ("userreal3",   ir->userreal3,  0);
2304      RTYPE ("userreal4",   ir->userreal4,  0);
2305  #undef CTYPE
2306
2307      // Brad defined file read
2308      //
2309      //
2310      FILE *stream;
2311      stream = fopen(is->userstr1,"r");
2312      int line_len = 100000;
2313      char temp_str[line_len];
2314      char indctr[1];
2315      int indices[2];
2316      double grid_params[3];
2317      float flt_tmp;
2318      int str_offset;
2319      int cnt;
2320      double shape_pot_scale  = 1.0;
2321      double offset_pot_scale = 1.0;
2322
2323      fgets(temp_str,line_len,stream);
2324  // Get the first character, should be a flag (u,n,p,d). In
         this case, u for the scaling factor.
2325      sscanf(temp_str,"%s",indctr);
2326      if (strcmp(indctr,"u")==0) {
2327          sscanf(temp_str, "%s%lf%lf", indctr, &
                 shape_pot_scale, &offset_pot_scale);
2328          fgets(temp_str,line_len,stream);
2329      }
2330      double shape_pot_scale_kt = 0.4*shape_pot_scale;
2331      (ir->pot_scale)[0] = shape_pot_scale;
2332      double offset_pot_scale_kt = 0.4*offset_pot_scale;
2333      (ir->pot_scale)[1] = offset_pot_scale;
```

```
2334      fprintf(stderr, "\n\nThe scaling factor for the shape
              potential is %.3g kJ/mol: %.3g kT-nm/mol at 300K.\n
              ",
2335                shape_pot_scale, shape_pot_scale_kt);
2336      fprintf(stderr, "\nThe spring constant for the offset
              potential is %.3g kJ/mol-nm^2: %.3g kT/mol-nm^2 at
              300K.\n",
2337                offset_pot_scale, offset_pot_scale_kt);
2338
2339      sscanf(temp_str,"%s",indctr);
2340
2341      int ndx_pairs = 1;
2342      while(strcmp(indctr,"n")==0) {
2343          sscanf(temp_str,"%s%i%i",indctr,&indices[0],&
                 indices[1]);
2344          (ir->pot_indices)[2*ndx_pairs-1] = indices[0];
2345          (ir->pot_indices)[2*ndx_pairs]   = indices[1];
2346          fgets(temp_str,line_len,stream);
2347          sscanf(temp_str,"%s",indctr);
2348          ndx_pairs += 1;
2349      }
2350      fprintf(stderr,"\nThe number of molecules being forced
              is %i.\n",ndx_pairs-1);
2351      if (ndx_pairs > 6)
2352      {
2353          fprintf(stderr,"\nThere are too many molecules,
                 only 5 density slots are allotted.\n");
2354      }
2355      (ir->pot_indices)[0] = (float) ndx_pairs-1;
2356
2357      sscanf(temp_str,"%s%lf%lf%lf",indctr,&grid_params[0],&
              grid_params[1],&grid_params[2]);
2358      (ir->pot_params)[0] = grid_params[0];
2359      (ir->pot_params)[1] = grid_params[1];
2360      (ir->pot_params)[2] = grid_params[2];
2361      fprintf(stderr,"\nThe potential runs from %.5gnm to %.5
              gnm in %.5gnm steps.\n",grid_params[0],grid_params
              [1],grid_params[2]);
2362      int num_points = round( (grid_params[1] - grid_params
              [0])/grid_params[2]+1 );
2363      fprintf(stderr,"\nThe number of points in the density
              array from parameter specification is %i.\n",
              num_points);
2364
2365      fgets(temp_str,line_len,stream);
```

```
2366        int neg_dens = 0;
2367
2368        sscanf(temp_str,"%s%n",indctr,&str_offset);
2369        memmove(temp_str, temp_str+str_offset,line_len-
               str_offset);
2370        memset(temp_str+line_len-str_offset,'0',str_offset);
2371        cnt = 0;
2372        while (sscanf(temp_str,"%f%n",&flt_tmp,&str_offset)>0)
               {
2373            (ir->exp_dens)[cnt] = flt_tmp;
2374            cnt+=1;
2375            if (flt_tmp < 0) {
2376                neg_dens = 1;
2377            }
2378            memmove(temp_str, temp_str+str_offset,line_len-
                   str_offset);
2379            memset(temp_str+line_len-str_offset,'0',str_offset)
                   ;
2380        }
2381        if (neg_dens==1) {
2382            fprintf(stderr,"\n\n\nCRITICAL ERROR: Densities
                   less than zero encountered! Density should be
                   greater than or equal to zero. Please make and
                   adjustment!\n\n\n");
2383        }
2384
2385        fprintf(stderr,"\nThe number of points from reading the
               experimental density is %i.\n",cnt);
2386        if (cnt != num_points) {
2387            fprintf(stderr,"\n\n\nERROR: The number of points
                   in the density do not match the parameter
                   specification for the density, please make an
                   adjustment.\n");
2388        }
2389        fprintf(stderr,"\n- Brad\n\n");
2390
2391        // Initialize rest of potential parameters
2392        real brad_sum = 0.0;
2393        int cnt2;
2394        for (cnt2=0;cnt2<cnt;cnt2++)
2395        {
2396            brad_sum += grid_params[2]*(grid_params[0]+cnt2*
                   grid_params[2])*(ir->exp_dens)[cnt2];
2397        }
2398        (ir->exp_mean)[0] = brad_sum;
```

```
2399
2400        for (cnt2=0;cnt2<5;cnt2++)
2401        {
2402            for (cnt=0;cnt<10000;cnt++)
2403            {
2404                (ir->sim_dens)[cnt2][cnt] = 0.0;
2405            }
2406            (ir->sim_mean)[cnt2] = 0.0;
2407        }
2408        (ir->z_bbox) = 0.0;
2409
2410        // Second Moment of Experimental Potential, used to
               determine condition for inclusion in ensemble
2411        brad_sum = 0.0;
2412        for (cnt=0;cnt<10000;cnt++)
2413        {
2414            brad_sum += grid_params[2] * (ir->exp_dens)[cnt] *
                   pow((grid_params[0]+cnt*grid_params[2] - (ir->
                   exp_mean)[0]), 2.0);
2415        }
2416        ir->second_moment = pow(brad_sum, 0.5);
2417    //
2418    //
2419    // Brad
```

A.1.3.3 sim_util.cpp

This file contains the calculation of the bias force and its addition to the other forces. The definition of the bias force is given by the function **do_user_external_force**, located between **do_force_cutsGROUP** and **do_force** around line 1974.

```
1974    // Neutron derived forces - Brad
1975    //
1976    //
1977
1978    void do_user_external_force(rvec x[], rvec f[], real
           indices[], real params[],
1979                                real scale[], real exp_dens[],
                                       real exp_mean[],
1980                                real sim_dens[5][10000], real
                                       sim_mean[], double z_bbox,
1981                                double second_moment, t_mdatoms
                                        *mdatoms, t_commrec *cr)
1982    {
1983        int         ii, i, ndx_l, ndx_h, indx;
```

```
1984        double      offset, f_com, z;
1985
1986        int         homenr = mdatoms->homenr;
1987
1988        /* Loop over all pairs of indices:
1989           indices[0] is the number of molecules being forced.
1990           After indices[0], there are that many pairs of
                  integers in 'indices' containing the start
1991           and stop index in the .gro (or global index+1) for
                  each molecule        */
1992        for (ii=0;(ii<(int) indices[0]);ii++)
1993        {
1994
1995            ndx_l  = (int) indices[2*ii+1] - 1; // Set the
                   bounds of global index for forced atoms
1996            ndx_h  = (int) indices[2*ii+2] - 1;
1997            offset = sim_mean[ii] - exp_mean[0]; // The offset
                   aligns the two densities for the potential
1998            if (abs(offset) < second_moment)
1999            {
2000                offset = 0.0; // If the offset is less than the
                       profile width, don't apply an offset force
                       and take
2001            }                 // the experimental density in
                   its actual location, no offset.
2002            //f_com  = -1.0*scale[1]*offset; // The force of
                   the offset, not scaled by system size.
                   Problematic
2003            f_com = -1.0*scale[1]*offset/(ndx_h-ndx_l+1); //
                   The force of the offset
2004
2005            for (i=0; i<homenr; i++)
2006            {
2007                if ((cr->dd->gatindex[i] >= ndx_l)&&(cr->dd->
                       gatindex[i] <= ndx_h))
2008                {
2009                    z = x[i][2];
2010                    // Check the pbc on the coordinate
2011                    if (z < params[0])
2012                    {
2013                        z += z_bbox;
2014                    }
2015                    // Simulation density contribution
2016                    indx = floor(( z - params[0] ) / params[2])
                           ; // sim_dens index
```

```
2017                    f[i][2] += scale[0]*(sim_dens[ii][indx]-
                            sim_dens[ii][indx+1]) / params[2];
2018                    // Experimental density contribution
2019                    indx = floor(( z - offset - params[0] ) /
                            params[2]); // exp_dens index
2020                    // Only try to access the exp density if
                            the index falls within the array
2021                    if ( (indx >= 0) && (indx < (params[1]-
                            params[0])/params[2]) )
2022                    {
2023                        f[i][2] += -1.0*scale[0]*(exp_dens[indx
                                ]-exp_dens[indx+1]) / params[2];
2024                    }
2025
2026                    // Offset force biasing toward experimental
                             mean
2027                    f[i][2] += f_com;
2028                }
2029            }
2030        }
2031    }
2032
2033    //
2034    //
2035    //
```

Additionally, the neutron force protocol is called during the **do_force** routine. It is the very last line of this routine.

```
2098        // Brad - do external forcing
2099        do_user_external_force(x, f, inputrec->pot_indices,
2100            inputrec->pot_params, inputrec->pot_scale,
2101            inputrec->exp_dens, inputrec->exp_mean,
2102            inputrec->sim_dens, inputrec->sim_mean,
2103            inputrec->z_bbox, inputrec->second_moment,
2104            mdatoms, cr);
2105        // Brad
```

A.1.3.4 sim_util.h

Header file for **sim_util.cpp**, function insertion around line 85

```
85  //Brad
86  void BRAD_global_stat(gmx_global_stat_t gs, t_commrec *cr,
87                        t_inputrec *inputrec, t_state *state_local,
88                        t_mdatoms *mdatoms, t_state *state_global);
```

```
89 //Brad
```

A.1.3.5 stat.cpp

This file defines an update to global parameters, e.g. the average simulation density. It involves collecting the coordinates from the various mpi processes and aggregating the data. The updater, which is a modification of **global_stat** is located around line 404.

```
404 // Brad
405 void BRAD_global_stat(gmx_global_stat_t gs, t_commrec *cr,
406                       t_inputrec *inputrec, t_state *state_local
                              ,
407                       t_mdatoms *mdatoms, t_state *state_global)
408 {
409     t_bin *rb;
410     rb   = gs->rb;
411
412     int isim_dens_temp = 0;
413     int homenr    = mdatoms->homenr;
414     int i=0, j=0;
415
416     // This routine copies all the data to be summed into
          one big buffer
417     // using the t_bin struct.
418
419     //%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%//
420     // Calculate the local density //
421     //%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%//
422
423     // Obtain the global density parameters (nm)
424     double zmin   = (inputrec->pot_params)[0];
425     double zstep  = (inputrec->pot_params)[2];
426     double z_bbox = (state_local->box)[2][2];
427     double second_moment = (inputrec->second_moment);
428     double z, nrm, sum;
429     int    z_low = 0, z_hi = 0;
430     int    ndx_l, ndx_h, ii;
431     //int num_of_states = (state_local->num_of_states);
432
433     //%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%//
434     // Update z-vector for bounding box //
435     //%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%//
436
```

```
437     (inputrec->z_bbox) = z_bbox;
438
439     /* Loop over all pairs of indices:
440        indices[0] is the number of molecules being forced.
441        After indices[0], there are that many pairs of
              integers in 'indices' containing the start
442        and stop index in the .gro (or global index+1) for
              each molecule        */
443     for (ii = 0;ii < (int) (inputrec->pot_indices)[0];ii++)
444     {
445         //printf("\n\n\nloop=%i,node=%i\n\n\n", ii, cr->
               nodeid);
446         ndx_l = (int) (inputrec->pot_indices)[2*ii+1] - 1;
               // Set the bounds of global index for forced
               atoms
447         ndx_h = (int) (inputrec->pot_indices)[2*ii+2] - 1;
448         nrm   = (ndx_h-ndx_l+1)*pow(2.0*M_PI,0.5)*0.1; // #
                of atoms * integral of single gaussian (sigma
               = 0.1 nm) gives a unit area for full profile as
                the norm
449
450
451         // Intialize the density variable
452         for (i=0; i<10000; i++)
453         {
454             (inputrec->sim_dens_temp)[i] = 0.0;
455         }
456
457         // Add up contributions
458         for (i=0; i<homenr; i++)
459         {
460             if ((cr->dd->gatindex[i] >= ndx_l)&&(cr->dd->
                   gatindex[i] <= ndx_h))
461             {
462                 z = (state_local->x)[i][2];
463                 // If the z-value is smaller than the
                       minimum in the potential, add the z-
                       value of the bounding box
464                 //     otherwise, the index for the array
                       could compute to be negative. Only
                       works for rectangular pbc.
465                 if (z<zmin)
466                 {
467                     z += z_bbox;
468                 }
```

```
469                 // The contribution comes from z +/- 3
                        sigma
470                 z_low = floor( ( z-0.3 - zmin ) / zstep );
471                 z_hi  = floor( ( z+0.3 - zmin ) / zstep );
472
473                 for(j=z_low;(j<z_hi);j++) // Add in all the
                        contribuitons to the density between
                        +/- 3 sigma
474                 {
475                     (inputrec->sim_dens_temp)[j] += exp
                            (-0.5*pow( z-(zmin + zstep*j) ,2)
                            /0.01)/nrm; // sigma**2 = 0.01 nm
                            **2
476                 }
477             }
478         }
479
480         //%%%%%%%%%%%%%%%%%%%%%//
481         // Sum over the nodes //
482         //%%%%%%%%%%%%%%%%%%%%%//
483
484         isim_dens_temp = add_binr(rb, 10000, inputrec->
                sim_dens_temp);
485         where();
486
487         sum_bin(rb, cr);
488         where();
489
490         extract_binr(rb, isim_dens_temp, 10000, inputrec->
                sim_dens_temp);
491         where();
492
493         //%%%%%%%%%%%%%%%%%%%%%//
494         // Calculate the mean //
495         //%%%%%%%%%%%%%%%%%%%%%//
496
497         sum = 0.0;
498         for (i=0;i<10000;i++)
499         {
500             z = zmin + zstep*i;
501             sum += zstep*z*(inputrec->sim_dens_temp)[i];
502         }
503         (inputrec->sim_mean)[ii] = sum;
504
505         //%%%%%%%%%%%%%%%%%%%%%%%%%%%%%//
```

```
506             // Set The Simulation Density //
507             //%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%//
508
509             // If the offset < the width of the profile, update
                    the ensemble
510             if ( std::abs( (inputrec->sim_mean)[ii] - (inputrec
                    ->exp_mean)[0] ) < second_moment)
511             //if (0.0 == 0.0)
512             {
513                 for (i=0; i<10000; i++)
514                 {
515                     //(state_local->ens_dens)[i] = (
                            num_of_states*(inputrec->ens_dens)[i] +
                             (inputrec->sim_dens_temp)[i]) / (
                            num_of_states + 1); // The local
                            ens_dens has not been allocated in
                            memory!!!!!
516                     (state_global->ens_dens)[i] = ((
                            state_global->num_of_states)*(
                            state_global->ens_dens)[i] + (inputrec
                            ->sim_dens_temp)[i]) / ((state_global->
                            num_of_states) + 1);
517                     (inputrec->sim_dens)[ii][i] = (state_global
                            ->ens_dens)[i];
518                 }
519                 //num_of_states += 1;
520                 (state_global->num_of_states) += 1;
521                 (state_local->num_of_states)  += 1;
522             }
523             else
524             {
525                 for (i=0; i<10000; i++)
526                 {
527                     (inputrec->sim_dens)[ii][i] = (inputrec->
                            sim_dens_temp)[i];
528                 }
529             }
530         }
531 }
532 // Brad
```

A.1.3.6 broadcaststructs.cpp

This file is involved in spreading out the computation among the nodes. Changes were made within **bcast_state** around line 254. The added line sends the size of structures to the non-master node, in order to allocate space, but doesn't send the data itself.

```
237 void bcast_state(const t_commrec *cr, t_state *state)
238 {
239     int      i, nnht, nnhtp;
240     gmx_bool bAlloc;
241
242     if (!PAR(cr) || (cr->nnodes - cr->npmenodes <= 1))
243     {
244         return;
245     }
246
247     /* Broadcasts the state sizes and flags from the master
            to all nodes
248      * in cr->mpi_comm_mygroup. The arrays are not
            broadcasted. */
249     block_bc(cr, state->natoms);
250     block_bc(cr, state->ngtc);
251     block_bc(cr, state->nnhpres);
252     block_bc(cr, state->nhchainlength);
253     block_bc(cr, state->flags);
254     block_bc(cr, state->num_of_states); // Brad
```

A.1.3.7 md_support.cpp

The first modification to this file includes a routine that prepares the nodes for a collective computation and then calls the averaging protocol **BRAD_global_stat**. This modification is inserted at around line 432.

```
432 //Brad
433 void BRAD_compute_globals(FILE *fplog, gmx_global_stat_t
        gstat, t_commrec *cr, t_inputrec *ir,
434                       t_forcerec *fr, gmx_ekindata_t *ekind,
435                       t_state *state, t_state *state_global,
                              t_mdatoms *mdatoms,
436                       t_nrnb *nrnb, t_vcm *vcm,
                             gmx_wallcycle_t wcycle,
437                       gmx_enerdata_t *enerd, tensor
                             force_vir, tensor shake_vir,
                             tensor total_vir,
```

```
438                         tensor pres, rvec mu_tot, gmx_constr_t
                                constr,
439                         struct gmx_signalling_t *gs, gmx_bool
                               bInterSimGS,
440                         matrix box, gmx_mtop_t *top_global,
441                         gmx_bool *bSumEkinhOld, int flags)
442 {
443
444     gmx::ArrayRef<real> signalBuffer = prepareSignalBuffer(
            gs);
445     if (PAR(cr))
446     {
447         wallcycle_start(wcycle, ewcMoveE);
448         BRAD_global_stat(gstat, cr, ir, state, mdatoms,
                state_global);
449         wallcycle_stop(wcycle, ewcMoveE);
450     }
451     handleSignals(gs, cr, bInterSimGS);
452
453 }
454 //Brad
```

The other modification is located in **set_state_entries**, involved in the setting of space allocation and flags in the **state** structure. The modification is made near line 755.

```
743     if (state->x == NULL)
744     {
745         snew(state->x, state->nalloc);
746     }
747     if (EI_DYNAMICS(ir->eI))
748     {
749         state->flags |= (1<<estV);
750         if (state->v == NULL)
751         {
752             snew(state->v, state->nalloc);
753         }
754     }
755     // Brad
756     state->flags |= (1<<estnum_of_states);
757     state->flags |= (1<<estens_dens);
758     if (state->ens_dens == NULL)
759     {
760         snew(state->ens_dens, 10000);
761     }
762     // Brad
```

A.1.3.8 md_support.h

Header file for **md_support.cpp**, statement inserted at the end of the file.

```
124 //Brad
125 void BRAD_compute_globals(FILE *fplog,
126     gmx_global_stat_t gstat, t_commrec *cr, t_inputrec *ir,
127     t_forcerec *fr, gmx_ekindata_t *ekind,
128     t_state *state, t_state *state_global,
129     t_mdatoms *mdatoms, t_nrnb *nrnb, t_vcm *vcm,
130     gmx_wallcycle_t wcycle, gmx_enerdata_t *enerd,
131     tensor force_vir, tensor shake_vir, tensor total_vir,
132     tensor pres, rvec mu_tot, gmx_constr_t constr,
133     struct gmx_signalling_t *gs, gmx_bool bInterSimGS,
134     matrix box, gmx_mtop_t *top_global,
135     gmx_bool *bSumEkinhOld, int flags);
136 //Brad
```

A.1.3.9 txtdump.c

This file is responsible for the GROMACS program **dump**, which outputs the contents of certain files into readable formats. The modification comes in the routine **pr_inputrec**, around line 1130.

```
1130         //Brad
1131         pr_reals_of_dim(fp, indent, "pot_indices", ir->
                 pot_indices,    15, 1);
1132         pr_reals_of_dim(fp, indent, "pot_params",   ir->
                 pot_params,     3, 1);
1133         pr_reals_of_dim(fp, indent, "pot_scale",     ir->
                 pot_scale,     2, 1);
1134         pr_reals_of_dim(fp, indent, "exp_dens",       ir->
                 exp_dens, 10000, 1);
1135         pr_reals_of_dim(fp, indent, "exp_mean",       ir->
                 exp_mean,     1, 1);
1136         pr_double(fp, indent, "z_bbox", ir->z_bbox);
1137         pr_double(fp, indent, "second_moment", ir->
                 second_moment);
1138         //Brad
```

A.1.3.10 checkpoint.cpp

This file is involved in the checkpointing of the simulation, for restart. The first modification is to a list of names to be printed in the checkpoint, adding

the **number of states** and **ensemble density** (time averaged density) at line 101.

```
96  const char *est_names[estNR] =
97  {
98      "FE-lambda",
99      "box", "box-rel", "box-v", "pres_prev",
100     "nosehoover-xi", "thermostat-integral",
101     "x", "v","number of states in ensemble", "ensemble
            density", "SDx", "CGp", "LD-rng", "LD-rng-i",//
            Brad
102     "disre_initf", "disre_rm3tav",
103     "orire_initf", "orire_Dtav",
104     "svir_prev", "nosehoover-vxi", "v_eta", "vol0", "
            nhpres_xi", "nhpres_vxi", "fvir_prev", "fep_state",
             "MC-rng", "MC-rng-i"
105 };
```

The other modification includes the calls to the routines that write the data into the checkpoint file, within **do_cpt_state**.

```
986                 case estX:       ret      = do_cpte_rvecs(
                        xd, cptpEST, i, sflags, state->natoms,
                        &state->x, list); break;
987                 case estV:       ret      = do_cpte_rvecs(
                        xd, cptpEST, i, sflags, state->natoms,
                        &state->v, list); break;
988                 case estnum_of_states: ret =do_cpte_int(xd,
                         cptpEST, i, sflags, &state->
                        num_of_states, list);  break; // Brad
989                 case estens_dens: ret     = do_cpte_doubles
                        (xd, cptpEST, i, sflags,10000, &state->
                        ens_dens, list); break;// Brad
990                 case estSDX:     ret      = do_cpte_rvecs(
                        xd, cptpEST, i, sflags, state->natoms,
                        &state->sd_X, list); break;
```

A.1.3.11 typedefs.c

This file contains the initialization of the **state** variable. Modifications were made to the routine **init_state**. If the simulation density array is initially allocated the incorrect length of memory, it must be set to null for the GROMACS allocation routine to work properly.

```
328     if (state->nalloc > 0)
```

```
329     {
330         snew(state->x, state->nalloc);
331         snew(state->v, state->nalloc);
332     }
333     else
334     {
335         state->x = NULL;
336         state->v = NULL;
337     }
338     state->sd_X = NULL;
339     state->cg_p = NULL;
340     zero_history(&state->hist);
341     zero_ekinstate(&state->ekinstate);
342     init_energyhistory(&state->enerhist);
343     init_df_history(&state->dfhist, nlambda);
344     init_swapstate(&state->swapstate);
345     state->ddp_count        = 0;
346     state->ddp_count_cg_gl  = 0;
347     state->cg_gl            = NULL;
348     state->cg_gl_nalloc     = 0;
349     // Brad
350     if (sizeof(state->ens_dens)/sizeof(state->ens_dens[0])
        != 10000)
351     {
352         state->ens_dens = NULL;
353     }
354     // Brad
```

A.1.3.12 state.h

File defining the structure **t_state**. The first modification is to add flags for the new variables in the structure, around line 62. The added flags are **estnum_of_states** and **estens_dens**.

```
59  enum {
60      estLAMBDA,
61      estBOX, estBOX_REL, estBOXV, estPRES_PREV, estNH_XI,
            estTC_INT,
62      estX,   estV, estnum_of_states, estens_dens, estSDX,
            estCGP, estLD_RNG, estLD_RNGI, //Brad
63      estDISRE_INITF, estDISRE_RM3TAV,
64      estORIRE_INITF, estORIRE_DTAV,
65      estSVIR_PREV, estNH_VXI, estVETA, estVOL0, estNHPRES_XI
            , estNHPRES_VXI, estFVIR_PREV,
66      estFEPSTATE, estMC_RNG, estMC_RNGI,
```

```
67      estNR
68  };
```

Additionally, added to the end of the definition of **t_state** are the number of states variable and the array tracking the simulation density average, around line 260.

```
254 int     ddp_count;        /* The DD partitioning count for
        this state   */
255 int     ddp_count_cg_gl; /* The DD part. count for index_gl
          */
256 int     ncg_gl;           /* The number of local charge
        groups             */
257 int     *cg_gl;            /* The global cg number of the
        local cgs          */
258 int     cg_gl_nalloc;     /* Allocation size of cg_gl;
                  */
259
260 int     num_of_states;    /* Brad - number of states in time
        average */
261 double *ens_dens;     /* Brad - time average of densities */
```

A.1.3.13 inputrec.h

File defining the structure **t_inputrec**. To the end of the structure were added input parameters associated with the potential, around line 440.

```
440     //Brad
441     real            pot_indices[15];
442     real            pot_params[3];
443     real            pot_scale[2];
444     real            exp_dens[10000];
445     real            exp_mean[1];
446     real            sim_dens[5][10000];
447     real            sim_dens_temp[10000];
448     real            sim_mean[5];
449     double          z_bbox;
450     double          second_moment;
451     //Brad
```

A.1.3.14 domdec.cpp

This file deals with the decomposition of the simulation box into domains and assigning atoms to them. The first modification is in **dd_collect_state**. Since there are no local copies of the added variables, they can be ignored.

```
1647        case estSDX:
1648            dd_collect_vec(dd, state_local, state_local->sd_X,
                    state->sd_X);
1649            break;
1650        case estCGP:
1651            dd_collect_vec(dd, state_local, state_local->cg_p,
                    state->cg_p);
1652            break;
1653        case estnum_of_states: //Brad
1654        case estens_dens: //Brad
1655        case estDISRE_INITF:
1656        case estDISRE_RM3TAV:
1657        case estORIRE_INITF:
1658        case estORIRE_DTAV:
1659            break;
```

The next function of interest is **dd_realloc_state**, where the same modifications are made.

```
1690        case estSDX:
1691            srenew(state->sd_X, state->nalloc);
1692            break;
1693        case estCGP:
1694            srenew(state->cg_p, state->nalloc);
1695            break;
1696        case estnum_of_states: //Brad
1697        case estens_dens: //Brad
1698        case estDISRE_INITF:
1699        case estDISRE_RM3TAV:
1700        case estORIRE_INITF:
1701        case estORIRE_DTAV:
1702            /* No reallocation required */
1703            break;
```

Now, in the function **dd_distribute_state**:

```
1952        case estCGP:
1953            dd_distribute_vec(dd, cgs, state->cg_p, state_local
                    ->cg_p);
1954            break;
1955        case estnum_of_states: //Brad
1956        case estens_dens: //Brad
```

Function **dd_redistribute_cg**:

```
4722            case estSDX: bSDX = (state->flags & (1<<i)); break;
4723            case estCGP: bCGP = (state->flags & (1<<i)); break;
4724            case estnum_of_states: //Brad
```

```
4725            case estens_dens: //Brad
```

Function **dd_sort_stae**:

```
9274            case estSDX:
9275                order_vec_atom(dd->ncg_home, cgindex, cgsort,
                       state->sd_X, vbuf);
9276                break;
9277            case estCGP:
9278                order_vec_atom(dd->ncg_home, cgindex, cgsort,
                       state->cg_p, vbuf);
9279                break;
9280            case estnum_of_states: //Brad
9281            case estens_dens: //Brad
```

A.1.3.15 md.cpp

This is the toplevel of the **mdrun** program. Just before the force call, the simulation density average is updated (currently hardcoded at 50 steps).

```
1068            // Brad
1069            if ((step%50)==0)
1070            {
1071                BRAD_compute_globals(fplog, gstat, cr, ir,
                       fr, ekind, state, state_global, mdatoms
                       , nrnb, vcm,
1072                            NULL, enerd, force_vir, shake_vir,
                                total_vir, pres, mu_tot,
1073                            constr, NULL, FALSE, state->box,
1074                            top_global, &bSumEkinhOld,
                                cglo_flags);
1075            }
1076            // Brad
```

A.2 Python

A.2.1 Class Structure

Density_Profile

- Protein_From_PXP (Derived Class)
- Protein_From_Configuration (Derived Class)
- Protein_From_Simulation (Derived Class)
- Bilayer_Profile (Derived Class)
- Density_Profile.convert_units()

Protein_From_PXP

- Protein_From_PXP()
- Protein_From_PXP.import_pxp_file()

Protein_From_Configuration

- Protein_From_Configuration()
- Protein_From_Configuration.import_configuration_file()

Protein_From_Simulation

- Protein_From_Simulation()
- Protein_From_Simulation.calculate_simulation_density()
- Protein_From_Simulation.import_simulation_density_from_atom_selection()
- Protein_From_Simulation.import_simulation_density_from_trajectory()
- Protein_From_Simulation.average_density_over_trajectory()

Bilayer_Profile

- Bilayer_From_PXP (Derived Class)
- Bilayer_From_Simulation (Derived Class)

Bilayer_From_PXP

Bilayer_From_PXP()

Bilayer_From_PXP.import_pxp_file()

Bilayer_From_Simulation

Bilayer_From_Simulation()

Bilayer_From_Simulation.calculate_simulation_density()

Bilayer_From_Simulation.import_simulation_density_from_universe()

Bilayer_From_Simulation.import_simulation_density_from_trajectory()

Helper Functions And Associated Classes

Profile_Comparison

compare_simulation_to_reference()

write_configuration_file()

Plotting

Plot_Densities()

A.2.2 Module Description

Density_Profile

Toplevel class with three derived classes. During normal operations, there should be no need to construct an instance of this class, only its derived classes. Density_Profile has one function that is inherited by the derived classes.

Density_Profile.convert_units (length_scale_factor, new_units)
Scales length dependent parameters of the class.
Parameters:

length_scale_factor: ***float***
The multiplicative factor that takes the current units to the new units. For example, if the units prior to conversion are *nm* and after conversion the new units are to be $\mathring{A}$, then the proper factor is 10.

new_units: ***character string***
Units after conversion. Throughout the code *Angstroms* are represented by *A* and *nanometers* are represented by *nm*.

Protein_From_PXP

Derived class of **Protein_Profile** representing a protein profile taken from the output of a fit to the neutron scattering data.

Protein_From_PXP (PXP_File=*None*, data_column_title = *'median_area'*, msigma_column_title = *'msigma'*, psigma_column_title = *'psigma'*, units = *'A'*)

Instantiation of the class. If a file name is supplied, the data will be read into the class using Protein_From_PXP.import_pxp_file().

Parameters:

PXP_File: ***character string, optional***
Name of file containing relevant fit data.

data_column_title: ***character string, optional***
Name for the column representing the protein profile in data file.

msigma_column_title: ***character string, optional***
Name for the column representing the lower bound of the confidence interval in the data file.

psigma_column_title: ***character string, optional***
Name for the column representing the upper bound of the confidence interval in the data file.

units: ***character string, optional***
Length units used in data file.

Returns: The instance of the class constructed.

Sets:

Protein_From_PXP.filename=filename *(if supplied)*
Protein_From_PXP.data_column_title=data_column_title
Protein_From_PXP.msigma_column_title=msigma_column_title
Protein_From_PXP.psigma_column_title=psigma_column_title
Protein_From_PXP.units=units

Protein_From_PXP.import_pxp_file (filename=*None*, data_column_title=*None*, msigma_column_title=*None*, psigma_column_title=*None*, units=*'A'*, include_confidence=*True*)

Imports data from the neutron fit file, defaulting to the class variables where necessary if arguments are not supplied (see class instantiation). If arguments are supplied, the class variables are overwritten.

Parameters:

PXP_File: ***character string, optional***
Name of file containing relevant fit data.

data_column_title: ***character string, optional***
Name for the column representing the protein profile in data file.

msigma_column_title: ***character string, optional***
Name for the column representing the lower bound of the confidence interval in the data file.

psigma_column_title: ***character string, optional***
Name for the column representing the upper bound of the confidence interval in the data file.

units: ***character string, optional***
Length units used in data file.

include_confidence: ***boolean, optional***
Indicates whether or not to import the confidence intervals.

Sets:

Protein_From_PXP.zmin: ***float***
Start of the z-axis.

Protein_From_PXP.zmax: ***float***
End of the z-axis.

Protein_From_PXP.zstep: ***float***
Interval between points along the z-axis.

Protein_From_PXP.density: ***numpy array***
Protein median profile.

Protein_From_PXP.msigma: ***numpy array***
Lower bound of confidence interval at each point.

Protein_From_PXP.psigma: ***numpy array***
Upper bound of confidence interval at each point.

Protein_From_PXP.norm: ***float***
Integrated area under median profile. The profile and confidence bounds are normalized using this area so, to reconstruct the data in the file, the must be scaled by this number.

Protein_From_PXP.mean: ***float***
Center, $\mu^{(1)}$, of the median profile.

Protein_From_PXP.second_moment: ***float***
Width, $\mu^{(2)}$, of the median profile.

Protein_From_Configuration

Derived class of **Protein_Profile** representing a protein profile taken from the configuration file supplied to GROMACS to import the neutron data and potential parameters.

Protein_From_Configuration (configuration_file = *None*, units = *'nm'*)

Instantiation of the class. If a filename is supplied, the configuration file is read using Protein_From_Configuration.import_configuration_file()

Parameters:

configuration_file: ***character string, optional***

Configuration file name.

units: ***character string, optional***

Units of length used in configuration file, should be *nm* since these are the units GROMACS uses.

Returns: Class instance.

Sets:

Protein_From_Configuration.filename = configuration_file

Protein_From_Configuration.units = units

Protein_From_Configuration.import_configuration_file (filename = *None*, units = *'nm'*)

Imports data from the configuration file, defaulting to the class variables where necessary if arguments are not supplied. If arguments are supplied, the class variables are overwritten.

Parameters:

configuration_file: ***character string, optional***

Configuration file name.

units: ***character string, optional***

Units of length used in configuration file, should be *nm* since these are the units GROMACS uses.

Sets:

Protein_From_Configuration.atom_groups: ***numpy array (n×2)***

Each of the n pairs is a set of indices for a range of indices, corresponding to what atoms should be forced during simulation. An $(n \times 2)$ array was chosen to start building functionality for forcing multiple sets of atoms. This has not been implemented yet.

Protein_From_Configuration.potential_scaling: ***numpy array***
Contains two floats corresponding to the scaling factors, λ and k, of the potential.

Protein_From_Configuration.zmin: ***float***
Start of the z-axis.

Protein_From_Configuration.zmax: ***float***
End of the z-axis.

Protein_From_Configuration.zstep: ***float***
Interval between points along the z-axis.

Protein_From_Configuration.density: ***numpy array***
Protein profile to be used as experimental target in biasing.

Protein_From_Configuration.norm: ***float***
Area of the density profile. If it is not 1, the configuration file is incorrect.

Protein_From_Configuration.units: ***character string***

Protein_From_Configuration.mean: ***float***
Center, $\mu^{(1)}$, of the profile.

Protein_From_Configuration.second_moment: ***float***
Width, $\mu^{(2)}$, of the profile.

Protein_From_Configuration.radii: ***numpy array***
Radii of the atoms used in calculating their individual volume contribution to simulation densities. Default is for every atom to be set to $1\mathring{A}$. Still in testing.

Protein_From_Configuration.volume: ***float***
Sum of the cubed radii. If scaled by $4\pi/3$, representative of the total protein volume if each atom had the radius in *Proten_From_Configuration.radii*. Used in calculating the normalization in simulation densities.

Protein_From_Simulation

Derived class of **Protein_Profile** representing a protein profile calculated from a simulation trajectory. The class can be supplied with an instance of *Protein_From_Configuration* to use as a reference, both to define the z-axis of the density and to later use in calculating differences between the profiles.

Protein_From_Simulation (structure_file = *None*, trajectory_file = *None*, atomselection = *None*, reference_profile_from_configuration = *None*, units = *'A'*)

Instantiation of the class. The trajectory is not automatically imported for calculation of density.

Parameters:

structure_file: ***character string, optional***
Structre file for simulation. ('.gro')

trajectory_file: ***character string, optional***
Trajectory file for simulation. ('.trr', '.xtc')

atomselection: ***MDAnalysis.core.groups.AtomGroup, optional***
Instead of reading in a trajectory, the densities can be calculated using this supplied selection of atoms from the desired trajectories.

reference_profile_from_configuration: ***Protein_From_Configuration, optional***
A reference profile that enables automated selection of certain parameters for the density, to provide consistency with the supplied configuration file.

units: ***character string***
MDAnalysis uses Angstroms by default. It converts from the native nanometer units used in GROMACS.

Returns: Class instance.

Sets:

Protein_From_Simulation.structure_file = structure_file
Protein_From_Simulation.trajectory_file = trajectory_file
Protein_From_Simulation.atomselection = atomselection
Protein_From_Simulation.reference_profile = reference_profile_from_configuration
Protein_From_Simulation.units = units

Protein_From_Simulation.calculate_simulation_density ()

Calculates the density from simulation at everypoint in the trajectory referred to by class variables. The range of the z-axis for the density and other parameters necessary to construct the density are determined by the values of the class variables. If all the proper class variables are not set before calling this method, it is recomended that *import_simulation_density_from_atom_selection* or *import_simulation_density_from_trajectory* be used instead, since they call this method as a subroutine.

Sets:

Protein_From_Simulation.density_trajectory:
list of numpy arrays
Each element is the density profile at the corresponding time step in *Protein_From_Simulation.frames*

Protein_From_Simulation.tracking_of_mean: ***list of floats***
Center, $\mu^{(1)}$, of each profile in *Protein_From_Simulation.density_trajectory*

Protein_From_Simulation.tracking_of_second_moment: ***list of floats***
Width, $\mu^{(2)}$, of each profile in *Protein_From_Simulation.density_trajectory*

Protein_From_Simulation.density: ***numpy array***
Average density profile density over frames examined.

Protein_From_Simulation.mean: ***float***
Center, $\mu^{(1)}$, of *Protein_From_Simulation.density*

Protein_From_Simulation.second_moment: ***float***
Width, $\mu^{(2)}$, of *Protein_From_Simulation.density*

Protein_From_Simulation.import_simulation_density_from_atom_selection (atomselection = *None*, frames = *None*, reference_profile_from_configuration = *None*, units = *'A'*)

Uses an MDAnalysis selection of atoms to construct the density over the course of an MD trajectory. Calls method *Protein_From_Simulation.calculate_simulation_density()*

Parameters:

atomselection: ***MDAnalysis.core.groups.AtomGroup, optional***
Group of atoms from which to calculate density. This can also be set via *Protein_From_Simulation.atomselection*

frames: ***list of integers, optional***
This specifies which frames in the trajectory associated with *atomselection* to use when examining the density. If not supplied and not set

in *Protein_From_Simulation.frames*, the method will prompt the user to enter a list of frames.

reference_profile_from_configuration: ***Protein_From_Configuration, optional***

A reference profile that enables automated selection of certain parameters for the density, to provide consistency with the supplied configuration file. If none is provided or stored in *Protein_From_Simulation.reference_profile*, some properties of the density will be requested of the user via prompt.

units: ***character string, optional***

MDAnalysis default is *'A' (Angstroms)*.

Protein_From_Simulation.import_simulation_density_from_trajectory
(structure_file = *None*, trajectory_file = *None*, frames = *None*, reference_profile_from_configuration = *None*, units = *'A'*)
Constructs the density of a set of atoms from an MD trajectory. Calls method *Protein_From_Simulation.calculate_simulation_density()*

Parameters:

structure_file: ***character string, optional***

MD structure file ('.gro'). If not provided or set in *Protein_From_Simulation.structure_file*, a prompt will ask for it as input.

trajectory_file: ***character string, optional***

MD trajectory file ('.trr', '.xtc'). If not provided or set in *Protein_From_Simulation.trajectory_file*, a prompt will ask for it as input.

frames: ***list of integers, optional***

This specifies which frames in the trajectory associated with *atomselection* to use when examining the density. If not supplied and not set in *Protein_From_Simulation.frames*, the method will prompt the user to enter a list of frames.

reference_profile_from_configuration: ***Protein_From_Configuration, optional***

A reference profile that enables automated selection of certain parameters for the density, to provide consistency with the supplied configuration file. If none is provided or stored in *Protein_From_Simulation.reference_profile*, some properties of the density will be requested of the user via prompt.

units: ***character string, optional***

MDAnalysis default is *'A' (Angstroms)*.

Protein_From_Simulation.average_density_over_trajectory (frame_subset = *None*)

Averages the density trajectory contained in *Protein_From_Simulation.density_trajectory* over the subset of its frames. For example, one might wish to only start tracking the density after a certain equilibration time.

Parameters:

frame_subset: ***list of integers, optional***

This specifies what frames to average over. The indices provided are relative to *Protein_From_Simulation.density_trajectory*, not the MD trajectory file. For example, if *Protein_From_Simulation.density_trajectory = [25, 26, 27, 28, ...]* and *frame_subset = [2,3,4, ...]*, then the frames of the MD trajectory that are used to average start at *27*. The default is to use all the frames in *Protein_From_Simulation.density_trajectory*.

Sets:

Protein_From_Simulation.frame_subset = frame_subset

Protein_From_Simulation.density

Protein_From_Simulation.mean

Protein_From_Simulation.second_moment

Bilayer_Profile

Derived class of **Density_Profile**, parent class for bilayer profiles from different sources.

Bilayer_Profile ()

Instantiation of the class. Constructs dictionary for picking out different subgroups of various lipids.

Sets:

lipid_groups_dictionary: ***dictionary***

A dictionary for lipids with entries that are dictionaries themselves for subgroups (*heads, tails, methyls*) that tell which atoms in each molecule belong to the various subgroups. Based on CHARMM forcefield for GROMACS.

Bilayer_From_PXP

Derived class of **Bilayer_Profile** representing a bilayer profile taken from the output of a fit to the neutron scattering data.

Bilayer_From_PXP (filename=*None*, protein_column_title = *'median_area'*, group_dictionary = {}, units = *'A'*, create_dictionary_interactively = *False*)

Instantiation of the class. If a file name is supplied, the data will be read into the class using Protein_From_PXP.import_pxp_file().

Parameters:

filename: ***character string, optional***

Name of file containing relevant fit data.

protein_column_title: ***character string, optional***

Name for the column representing the protein profile in data file.

group_dictionary: ***dictionary, optional***

Dictionary of subgroups that indicates which column names in the data file correspond to that particular subgroup. This can be defined interactively when importing the data.

units: ***character string, optional***

Length units used in data file.

create_dictionary_interactively: ***boolean, optional***

Whether or not to define *group_dictionary* using a rudimentary interface that can indicate what columns exist in the file.

Returns: The instance of the class constructed.

Sets:

Bilayer_From_PXP.filename=filename *(if supplied)*
Bilayer_From_PXP.protein_column_title=protein_column_title
Bilayer_From_PXP.group_dictionary=group_dictionary
Bilayer_From_PXP.units=units

Bilayer_From_PXP.import_pxp_file (filename=*None*, protein_column_title=*None*, group_dictionary=*None*, units=*'A'*, create_group_dictionary_interactively=*False*)

Imports data from the neutron fit file, defaulting to the class variables where necessary if arguments are not supplied (see class instantiation). If arguments are supplied, the class variables are overwritten.

Parameters:

filename: ***character string, optional***
Name of file containing relevant fit data.

protein_column_title: ***character string, optional***
Name for the column representing the protein profile in data file.

group_dictionary: ***dictionary, optional***
Dictionary of subgroups that indicates which column names in the data file correspond to that particular subgroup. This can be defined interactively when importing the data.

units: ***character string, optional***
Length units used in data file.

create_dictionary_interactively: ***boolean, optional***
Whether or not to define *group_dictionary* using a rudimentary interface that can indicate what columns exist in the file.

Sets:

Any supplied parameters.

Bilayer_From_PXP.zmin: ***float***
Start of the z-axis.

Bilayer_From_PXP.zmax: ***float***
End of the z-axis.

Bilayer_From_PXP.zstep: ***float***
Interval between points along the z-axis.

Bilayer_From_PXP.density_dictionary: ***dictionary referencing numpy arrays***
Dictionary matching subgroups to their density profiles.

Bilayer_From_PXP.protein_norm: ***float***
Area of protein median profile. This permits relative normalization when plotting bilayer and normalized protein together.

Bilayer_From_PXP.bilayer_center: ***float***
Center, $\mu^{(1)}$, of the *'tails'* subgroup's profile. This may be slightly different from the bilayer center if the upper and lower leaflet tail distributions are not fully symmetric.

Bilayer_From_Simulation

Derived class of **Bilayer_Profile** representing a bilayer profile taken from an MD simulation.

Bilayer_From_Simulation (structure_file = *None*, trajectory_file = *None*, universe = *None*, frames = *None*, lipid_resname_dictionary = {*'CHOL':'CHL1', 'DOPC':'DOPC', 'DOPS':'DOPS'*})
Instantiation of the class.

Parameters:

structure_file: ***character string, optional***
Name of simulation structure file ('.gro').

trajectory_file: ***character string, optional***
Name of simulation trajectory file ('.trr').

universe: ***MDAnalysis.core.universe, optional***
An alternative to the simulation structure file and trajectory file, the universe obtained by loading them using MDAnalysis.

frames: ***list of integers, optional***
What frames in the trajectory to use.

lipid_resname_dictionary: ***dictionary, optional***
Dictionary connecting the molecule names in *lipid_groups_dictionary* to the residue names in the MD files.

Returns: An instance of the class constructed.

Sets:

Bilayer_From_Simulation.structure_file = structure_file
Bilayer_From_Simulation.trajectory_file = trajectory_file
Bilayer_From_Simulation.universe = universe
Bilayer_From_Simulation.frames = frames
Bilayer_From_Simulation.resname_dict = lipid_resname_dictionary

---

Bilayer_From_Simulation.calculate_simulation_density (

bilayer_selection_dictionary, frames)
Calculates the average bilayer density over the frames supplied, using *bilayer_selection_dictionary* to ascribe atom selections to each subgroup of the bilayer for which density is to be calculated.

Parameters:

bilayer_selection_dictionary: ***dictionary***
The keys are the subgroups of the bilayer (*'heads', 'tails', 'methyls'*) and the values are atom selections for the MD universe that corresponds to those subgroups.

frames: ***list of integers***
What frames in the trajectory to use in calculating the average profile.

Sets:

Bilayer_From_Simulation.density_dictionary: ***dictionary***
A dictionary that has the same keys as *bilayer_selection_dictionary* and values that are the profiles for the respective keys, in the form of *numpy arrays*.

Bilayer_From_Simulation.total_density_norm ***float***
Total area under bilayer profile.

Bilayer_From_Simulation.bilayer_center ***float***
Center, $\mu^{(1)}$, of the total profile (all components included).

Bilayer_From_Simulation.bilayer_atomgroup_dictionary ***dictionary***
Copy of *bilayer_selection_dictionary* for reference.

Bilayer_From_Simulation.frames ***list of integers***
List of frames analyzed.

Bilayer_From_Simulation.import_simulation_density_from_universe (universe = *None*, frames = *None*, lipid_resname_dictionary = *None*) Takes an MDAnalysis universe containing a bilayer and calls *Bilayer_From_Simulation.calculate_simulation_density()* to calculate its density. Will use class attributes if no arguments are supplied.

Parameters:

universe: ***MDAnalysis.core.universe, optional***
An alternative to the simulation structure file and trajectory file, the universe obtained by loading them using MDAnalysis.

frames: ***list of integers, optional***
What frames in the trajectory to use.

lipid_resname_dictionary: ***dictionary, optional***
Dictionary connecting the molecule names in *lipid_groups_dictionary* to the residue names in the MD files.

Sets:

Bilayer_From_Simulation.structure_file: ***character string***
Structure file associated with the universe supplied.

Bilayer_From_Simulation.trajectory_file: ***character string***
Trajectory file associated with the universe supplied.

Bilayer_From_Simulation.zmin: ***Start of the z-axis.***

Bilayer_From_Simulation.zmax: ***float***
End of the z-axis.

Bilayer_From_Simulation.zstep: ***float***
Interval between points along the z-axis.

Bilayer_From_Simulation.import_simulation_density_from_trajectory
(structure_file = *None*, trajectory_file = *None*, frames = *None*, lipid_resname_dictionary = *None*)
Imports a trajectory from the supplied files into an MDAnalysis universe and calls *Bilayer_From_Simulation.import_simulation_density_from_universe()* to calculate its bilayer density. Will use class attributes if no arguments are supplied.

Parameters:

structure_file: ***character string, optional***
Name of simulation structure file ('.gro').

trajectory_file: ***character string, optional***
Name of simulation trajectory file ('.trr').

frames: ***list of integers, optional***
What frames in the trajectory to use.

lipid_resname_dictionary: ***dictionary, optional***
Dictionary connecting the molecule names in *lipid_groups_dictionary* to the residue names in the MD files.

Plotting (plotting.py)

Plot_Densities (List_of_Profiles, bilayer_color_dictionary = {})
Plots a group of density profiles.

Parameters:

List_of_Profiles: ***list of Density_Profiles or derived classes***
The profiles to be plotted.

bilayer_color_dictionary: ***dictionary, optional***
The keys of the dictionary are the keys to *density_dictionary* for any bilayers in the list of profiles. The values of the dictionary are color specifications for *matplotlib* to use when plotting the associated density.

Returns:

ax: ***matplotlib.axes._subplots.AxesSubplot***
The axis containing the plot.

plot_data: ***list***
This list contains each *matplotlib.lines.Line2D* instance created when a profile is plotted without confidence intervals and each *matplotlib.collections.PolyCollection* instance created when a profile is plotted with error bars.

Helper Functions and Associated Classes (functions.py)
class: Profile_Comparison

Class Attributes:

rmsd_inst: ***numpy array***

Root mean square difference for individual frames in trajectory.

rmsd_avg: ***numpy array***

Root mean square difference for average of all frames prior to current.

---

compare_simulation_to_reference (Simulation_Profile, Reference_Profile = *None*, Subset_Of_Frames_To_Look_At = *None*)

Calculates the root mean square differnce between a simulation profile and a reference profile. The average is over the z-axis.

Parameters:

Simulation_Profile: ***Protein_From_Simulation***

Simulation profile to be compared.

Reference_Profile: ***Protein_From_Configuration, optional***

Profile to be compared against the simulation. Reference can be omitted by setting *Protein_From_Simulation.reference_profile* in the supplied simulation profile.

Subset_Of_Frames_To_Look_At: ***list of integers, optional***

Set of indices that select the frames from *Protein_From_Simulations.frames* to examine. As in *Protein_From_Simulations.average_density_over_trajectory*, the indices refer to the list of frames in the simulation profile not the frames in the MD trajectory. Default is all frames.

Returns: ***Profile_Comparison instance***

---

write_configuration_file (Profile, filename, use_radii = *False*)

Creates a configuration file that supplies GROMACS with the necessary information to construct the biasing potential.

Parameters:

Simulation_Profile: ***Protein_From_PXP*** **or** ***Profile_From_Configuration***

Profile to be written into a configuration file. The units of length for all attributes should be nanometers. If not, an error will be raised.

filename: ***character string***

File name for the output.

use_radii: ***boolean***

Whether or not to write the radii for volume calculation that differentiates atom species. Not yet used, set to *False* by default.

---

prepare_neutron_profile_for_configuration (protein_profile, Bilayer_Profile_Neutron, Bilayer_Profile_MD)

Normalizes the protein profile and adjusts the z-axis to correspond with the simulation coordinates. The alignment is performed by caluclating the offset of the bilayer centers and modifying as is necessary. The protein profile is also truncated to ten *zsteps* above and below the nonzero densities.

Parameters:

protein_profile: ***Protein_From_PXP***

Profile to be written into a configuration file. The units of length for all attributes should be nanometers. If not, an error will be raised.

Bilayer_Profile_Neutron: ***Bilayer_Profile***

Bilayer profile from the same '.pxp' as the protein.

Bilayer_Profile_MD: ***Bilayer_Profile***

Bilayer profile from simulation configuration prior to bias simulation. This can be constructed from an average of a pre-bias trajectory or the '.gro' structure of the system input to the bias simulation.

Appendix B

Appendix

B.1 Jensen's Inequality And The KL Divergence

Jensen's inequality states that if $p(x)$ is a probability density for the random variable X and $f(x)$ is a convex function, then

$$\langle f(X) \rangle \geq f\big(\langle X \rangle \big) \tag{B.1}$$

Taking a random variable $Y = q(x)/p(x)$, for some function q, and the convex function $f(x) = -\log(x)$, Jensen's inequality states

$$\begin{aligned} \langle -\log(Y) \rangle &\geq -\log\big(\langle Y \rangle\big) \\ \int dx\ p(x) \left[-\log\left(\frac{q(x)}{p(x)}\right)\right] &\geq -\log\left(\int dx\ p(x) \frac{q(x)}{p(x)}\right) \\ -\int dx\ p(x) \log\left(\frac{q(x)}{p(x)}\right) &\geq -\log\left(\int dx\ q(x)\right) \end{aligned} \tag{B.2}$$

If $q(x)$ is another probability distribution, then the KL divergence with respect to $q(x)$ satisfies the following

$$\begin{aligned}\eta\left[p(x)||q(x)\right] &= \int dx\ p(x)\log\left(\frac{p(x)}{q(x)}\right)\\ &= -\int dx\ p(x)\log\left(\frac{q(x)}{p(x)}\right)\\ &\geq -\log\left(\int dx\ q(x)\right) = 0\end{aligned} \tag{B.3}$$

B.2 Random Variable Transform For Sums Of Symmetric Distributions

Given two independent random variables $\mathcal{X}$ and $\mathcal{Y}$, with associated probability density fucntion $P_{\mathcal{X}}(x)$ and $P_{\mathcal{Y}}(y)$ for $x \in \mathcal{X}$ and $y \in \mathcal{Y}$, a third random variable constructed from their sum, $\mathcal{Z} = \mathcal{X} + \mathcal{Y}$, will have a probability density function given by the Random Variable Transformation (RVT) Theorem.[101]

$$P_{\mathcal{Z}}(z) = \iint dx\,dy\ P_{\mathcal{X}}(x)P_{\mathcal{Y}}(y)\delta\big(z-(x+y)\big) \tag{B.4}$$

If $P_{\mathcal{X}}$ and $P_{\mathcal{Y}}$ are zero mean, symmetric distributions then

$$\begin{aligned}P_{\mathcal{Z}}(z) &= \iint dx\,dy\ P_{\mathcal{X}}(-x)P_{\mathcal{Y}}(-y)\delta\big(z-(x+y)\big)\\ \text{Substituting:}\quad & x=-u \quad y=-v\\ &= \iint (-du)(-dv)\ P_{\mathcal{X}}(u)P_{\mathcal{Y}}(v)\delta\big(z-(-u-v)\big)\\ &= \iint du\,dv\ P_{\mathcal{X}}(u)P_{\mathcal{Y}}(v)\delta\big(-z-(u+v)\big)\\ &= P_{\mathcal{Z}}(-z)\end{aligned} \tag{B.5}$$

$P_{\mathcal{Z}}$ is also a zero mean, symmetric distribution.

B.3 Markov Chain Monte Carlo Methods For Simulating Restrained Ensemble On A One Dimensional Potential

For a restrained ensemble with N replicas, the vector of random variables to be chained includes the postions of each replica's particle. These particles are updated at each step via the Metropolis algorithm

$$q_{i,t+1} = q_{i,t} + r_i \tag{B.6}$$

where the r_i are drawn from a zero mean, symmetric distribution. In addition, the average position of all the particles at each time step must be equal to the predetermined restraint value.

$$\sum_{i}^{N} q_{i,t} = Q \tag{B.7}$$

There are methods for parameter proposals in the presence of constraints, but it is not a difficult problem to construct N random numbers from an $N-1$ dimensional space for the problem at hand. To illustrate this, take the case of $N = 3$. Then eq (B.7) reduces to, for each step in time,

$$q_1 + q_2 + q_3 = 3Q \tag{B.8}$$

Eq (B.8) is the equation of a plane, which can be expressed as a vector equation and is visually represented in figure B.1.

$$\vec{q} \cdot \vec{n} = 3Q \tag{B.9}$$

Where $\vec{n}$ is the three-vector containing one in all its entries. This geometric description suggests a method for ensuring that proposals have the correct average, *i.e.* staying in the plane of solutions to eq (B.8). If at the current step the point already lies in the plane, then stepping along a linear combination of vectors in the plane will ensure that the next point is also in the plane since the vectors in the plane are orthogonal to vectors normal to the plane.

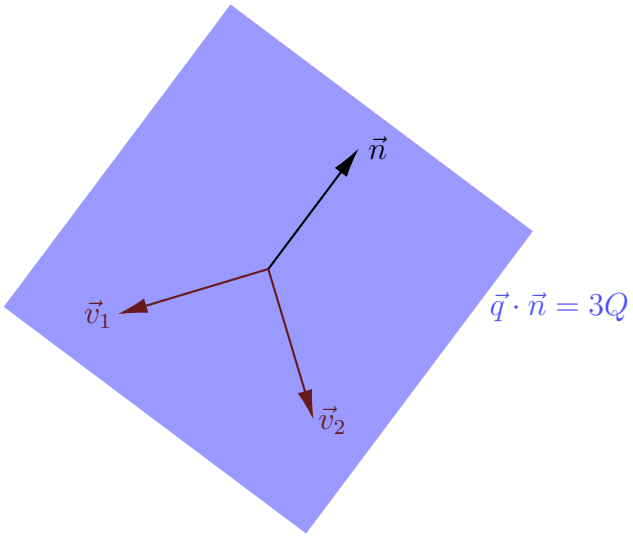

Figure B.1: Visual description of set of particle positions with fixed average for $N = 3$. $\vec{n}$ is a vector normal to the plane, with entries consisting exclusively of ones. $\vec{v}_1$ and $\vec{v}_2$ are vectors in the plane of solutions.

$$\vec{q}_{t+1} = \vec{q}_t + r_1\vec{v}_1 + r_2\vec{v}_2 \tag{B.10}$$

$$\begin{aligned} \vec{n} \cdot \vec{q}_{t+1} &= \vec{n} \cdot \vec{q}_t + r_1\vec{n} \cdot \vec{v}_1 + r_2\vec{n} \cdot \vec{v}_2 \\ &= \vec{n} \cdot \vec{q}_t \\ &= 3Q \end{aligned} \tag{B.11}$$

This method generalizes to N replicas easily, a set of orthonormal vectors must be constructed with the constraint that one of the vector's entries are entirely composed of $\frac{1}{\sqrt{N}}$ (the generalized $\vec{n}$ vector after normalization). The Gram-Schmidt process can be performed by starting with the normalized $\vec{n}$ and performing the algorithm on $N-1$ vectors with all zeros except for one entry. Since the original set is linearly independent, they are certain to span the entire N-dimensional space after the Gram-Schmidt process.

Since the generalized $\vec{n}$ is proportional to the first vector in the set, the rest of the vectors are orthogonal. This permits the modification of particle position vectors by linear combinations of these orthogonal vectors to produce

position vectors with the same average value, as shown in eq (B.13)

$$\vec{q}_{t+1} = \vec{q}_t + \sum_{i=2}^{N} r_i \vec{v}_i \tag{B.12}$$

$$\begin{aligned}
\langle \vec{q}_{t+1} \rangle &= \frac{1}{N} \sum_j q_{j,t+1} = \frac{1}{N} (\vec{n} \cdot \vec{q}_{t+1}) \\
&= \frac{1}{N} \left(\vec{n} \cdot \vec{q}_t + \sum_{i=2}^{N} r_i \vec{n} \cdot \vec{v}_i \right) \\
&= \frac{1}{N} (\vec{n} \cdot \vec{q}_t) = \langle \vec{q}_t \rangle
\end{aligned} \tag{B.13}$$

Using this formulation, parameter proposals can be constructed for simulation in which the mean of the parameters does not move. The r_i in eq (B.12) are the random weights in new parameter space– the $\vec{v}_i$ vectors, which number $N - 1$. The resulting random variables that generate the position proposal are sums of random variables themselves, weighted by the entries of the $\vec{v}_i$.

$$\begin{aligned}
q_{j,t+1} &= q_{j,t} + \sum_i r_i v_{j,i} \\
&= q_{j,t} + \alpha_j
\end{aligned} \tag{B.14}$$

Eq (B.14) shows the transormation between the random variables and, if the r_i are drawn from even probability density function, then so are the α_j. (See section B.2)

B.3.1 Python Code For Restrained Ensemble Simulation

```
1 import numpy as np
2 from matplotlib import pyplot as plt
3 import random
4
5 ###################################
6 # Define the particle potential #
```

```
7  ##################################
8
9  def Potential(q):
10     return 25*(q-0.25)**4 - q*np.cos(q)
           + (1.0/(q**2 + 0.5))*np.sin(20*q)
11
12 ##########################################################
13 # Method for calculating vectors orthonormal to normal #
14 ##########################################################
15 def Ortho_Set(n):
16     # This will produce unbiased simulations
17     if n==1:
18         return [np.array([1.0])]
19     # Otherwise find the desired vectors
20     vec1 = np.ones(n)/(n**0.5) # Proportional to normal
21     ret = []                   # List of vectors to return
22     # Gram-Schmidt
23     for i in range(n-1):
24         # Starting vectors have one nonzero entry
25         v = np.zeros(n)
26         v[i] = 1.0
27         # Subtract contribution of normal vector
28         u = v - np.dot(v,vec1)*vec1
29         # Subtract contributions from the vectors
30         #   that came before
31         for j in range(i):
32             u = u - np.dot(v,ret[j])*ret[j]
33         # Add the new vector, normalized
34         ret.append(u/(np.dot(u,u)**0.5))
35
36     return ret
37
38 ##########################
39 # Simulation Parameters #
40 ##########################
41 F = 100000000          # Samples across all particles
42 N = 200                # Number of particles
43 T = F/N                # Number of step in simulation
44 interval_size = 0.008 # By what to scale random interval
45 q_bias = -0.127        # Observed mean position
46 rejects = 0            # For tracking rejected proposals
47 qout = np.zeros((T,N))# Markov Chain Results
48 delta_q               # Tracking random changes in q
49 ortho_vecs = Ortho_Set(N) # Proposal Vectors
50
```

```
################################
# Initialize Parameter Vector #
################################

for i in range(N):
    qout[0,i] = random.uniform(q_bias-1.0, q_bias+1.0)
if N > 1:
    qout[0] -= (np.mean(qout[0]) - q_bias) # Adjust mean

############################
# Sample The Distribution #
############################

for i in range(1,T):
    ############
    # Proposal #
    ############
    qnew = np.array([j for j in qout[i-1]])
    for vec in ortho_vecs:
        tmp = interval_size*random.uniform(-1.0,1.0)*vec
        for j in range(len(tmp)):
            qnew[j] += tmp[j]
    delta_q[i-1] = qnew - qout[i-1]
    ##############
    # Acceptance #
    ##############
    Unew = sum([Potential(j) for j in qnew])
    Uold = sum([Potential(j) for j in qout[i-1]])
    if Unew < Uold:
        #################
        # Always accept #
        #################
        qout[i] = np.array([j for j in qnew])
    else:
        ################################################
        # Accept with probability e^(-(U_new - U_old)) #
        ################################################
        P = random.uniform(0,1.0)
        if P < np.exp(Uold - Unew):
            qout[i] = np.array([j for j in qnew])
        else:
            qout[i] = qout[i-1]
            rejects += 1
```

Bibliography

[1] G. Bussi, D. Donadio, and M. Parrinello, "Canonical sampling through velocity rescaling," *The Journal of Chemical Physics*, vol. 126, p. 014101, 2007.

[2] H. J. C. Berendsen, J. P. M. Postma, W. F. van Gunsteren, A. DiNola, and J. R. Haak, "Molecular dynamics with coupling to an external bath," *The Journal of Chemical Physics*, vol. 81, pp. 3684–3690, 1984.

[3] S. Nosé, "A molecular dynamics method for simulations in the canonical ensemble," *Molecular Physics*, vol. 52(2), pp. 255–268, 1984.

[4] W. G. Hoover, "Canonical dynamics: Equilibrium phase-space distributions," *Physical Review A*, vol. 31, pp. 1695–1697, 1985.

[5] H. C. Andersen, "Molecular dynamics simulations at constant pressure and/or temperature," *The Journal of Chemical Physics*, vol. 72, p. 2384, 1980.

[6] M. Parrinello and A. Rahman, "Polymorphic transitions in single crystals: A new molecular dynamics method," *Journal of Applied Physics*, vol. 52, pp. 7182–7190, 1981.

[7] G. J. Martyna, D. J. Tobias, and M. L. Klein, "Constant pressure molecular dynamics algorithms," *The Journal of Chemical Physics*, vol. 101(5), pp. 4177–4189, 1994.

[8] S. E. Feller, Y. Zhang, and R. W. Pastor, "Constant pressure molecular dynamics simulation: The langevin piston method," *The Journal of Chemical Physics*, vol. 103(11), pp. 4613–4621, 1995.

[9] J. E. Basconi and M. R. Shirts, "Effects of temperature control algorithms on transport properties and kinetics in molecular dynamics simulations," *Journal of Chemical Theory and Computation*, vol. 9(7), pp. 2887–2899, 2013.

[10] G. J. Martyna and M. L. Klein, "Nosé–hoover chains: The canonical ensemble via continuous dynamics," *The Journal of Chemical Physics*, vol. 97, pp. 2635–2643, 1992.

[11] A. Grossfield and D. M. Zuckerman, "Chapter 2 quantifying uncertainty and sampling quality in biomolecular simulations," *Annual Reports in Computational Chemistry*, vol. 5, pp. 23–48, 2009.

[12] A. Grossfield, S. E. Feller, and M. C. Pitman, "Convergence of molecular dynamics simulations of membrane proteins," *Proteins*, vol. 67(1), pp. 31–40, 2007.

[13] J. Yu, T. Ha, and K. Schulten, "Structure-based model of the stepping motor of PcrA helicase," *Biophysical Journal*, vol. 91, pp. 2097–2114, 2006.

[14] D. M. Zuckerman, "Equilibrium sampling in biomolecular simulations," *Annual Review of Biophysics*, vol. 40, pp. 41–62, 2011.

[15] R. H. Swendsen and J.-S. Wang, "Replica monte carlo simulation of spin-glasses," *Physical Review Letters*, vol. 57, pp. 2607–2609, 1986.

[16] Y. Sugita and Y. Okamoto, "Replica-exchange molecular dynamics method for protein folding," *Chemical Physics Letters*, vol. 314(1), pp. 141–151, 1999.

[17] G. A. Huber and J. A. McCammon, "Weighted-ensemble simulated annealing: Faster optimization on hierarchical energy surfaces," *Physical Review E*, vol. 55, pp. 4822–4825, 1997.

[18] R. M. Neal, "Annealed importance sampling," *Statistics and Computing*, vol. 11, pp. 125–139, 2001.

[19] E. Lyman and D. M. Zuckerman, "Annealed importance sampling of peptides," *The Journal of Chemical Physics*, vol. 127(6), p. 065101, 2007.

[20] J. Hritz and C. Oostenbrink, "Hamiltonian replica exchange molecular dynamics using soft-core interactions," *The Journal of Chemical Physics*, vol. 128(14), p. 144121, 2008.

[21] D. Hamelberg, J. Mongan, and J. A. McCammon, "Accelerated molecular dynamics: A promising and efficient simulation method for biomolecules," *The Journal of Chemical Physics*, vol. 120(24), p. 11919, 2004.

[22] T. Shen and D. Hamelberg, "A statistical analysis of the precision of reweighting-based simulations," *The Journal of Chemical Physics*, vol. 129(3), p. 034103, 2008.

[23] Y. Sugita, A. Kitao, and Y. Okamoto, "Multidimensional replica-exchange method for free-energy calculations," *The Journal of Chemical Physics*, vol. 113(15), pp. 6042–6051, 2000.

[24] O. Teleman and B. Jönsson, "Vectorizing a general purpose molecular dynamics simulation program," *Journal of Computational Chemistry*, vol. 7(1), pp. 58–66, 1986.

[25] M. Tuckerman, B. J. Berne, and G. J. Martyna, "Reversible multiple time scale molecular dynamics," *The Journal of Chemical Physics*, vol. 97(3), pp. 1990–2001, 1992.

[26] M. Christen and W. F. van Gunsteren, "Multigraining: An algorithm for simultaneous fine-grained and coarse-grained simulation of molecular systems," *The Journal of Chemical Physics*, vol. 124(15), p. 154106, 2006.

[27] P. Liu, Q. Shi, E. Lyman, and G. A. Voth, "Reconstructing atomistic detail for coarse-grained models with resolution exchange," *The Journal of Chemical Physics*, vol. 129(11), p. 114103, 2008.

[28] E. Lyman and D. M. Zuckerman, "Resolution exchange simulation with incremental coarsening," *Journal of Chemical Theory and Computation*, vol. 2(3), pp. 656–666, 2006.

[29] A. B. Mamonov, D. Bhatt, D. J. Cashman, Y. Ding, and D. M. Zuckerman, "General library-based monte carlo technique enables equilibrium

sampling of semi-atomistic protein models," *Journal of Chemical Theory and Computation*, vol. 113(31), pp. 10891–10904, 2009.

[30] R. M. Venable and R. W. Pastor, "Frictional models for stochastic simulations of proteins," *Biopolymers*, vol. 27(6), pp. 1001–1014, 1988.

[31] G. Lamm and A. Szabo, "Langevin modes of macromolecules," *The Journal of Chemical Physics*, vol. 85(12), pp. 7334–7348, 1986.

[32] J. A. McCammon, S. H. Northrup, M. Karplus, and R. M. Levy, "Helix-coil transitions in a simple polypeptide model," *Biopolymers*, vol. 19(11), pp. 2033–2045, 1980.

[33] C. L. Brooks, A. Brünger, and M. Karplus, "Active site dynamics in protein molecules: A stochastic boundary molecular-dynamics approach," *Biopolymers*, vol. 24(5), pp. 843–865, 1985.

[34] D. L. Ermak and J. A. McCammon, "Brownian dynamics with hydrodynamic interactions," *The Journal of Chemical Physics*, vol. 69(4), pp. 1352–, 1978.

[35] D. M. Zuckerman, "Simulation of an ensemble of conformational transitions in a united-residue model of calmodulin," *The Journal of Physical Chemistry B*, vol. 108(16), pp. 5127–5137, 2004.

[36] J. Shimada, E. L. Kussell, and E. I. Shakhnovich, "The folding thermodynamics and kinetics of crambin using an all-atom monte carlo simulation," *Journal of Molecular Biology*, vol. 308(1), pp. 79–95, 2001.

[37] W. L. Jorgensen and J. Tirado-Rives, "Monte carlo vs molecular dynamics for conformational sampling," *The Journal of Physical Chemistry*, vol. 100(34), pp. 14508–14513, 1996.

[38] U. Nowak, R. W. Chantrell, and E. C. Kennedy, "Monte carlo simulation with time step quantification in terms of langevin dynamics," *Physical Review Letters*, vol. 84(1), pp. 163–166, 2000.

[39] J. P. Ulmschneider, M. B. Ulmschneider, and A. D. Nola, "Monte carlo vs molecular dynamics for all-atom polypeptide folding simulations," *The Journal of Physical Chemisty B*, vol. 110(33), pp. 16733–16742, 2006.

[40] Y. Ding, A. B. Mamonov, and D. M. Zuckerman, "Efficient equilibrium sampling of all-atom peptides using library-based monte carlo," *The Journal of Physical Chemistry B*, vol. 114(17), pp. 5870–5877, 2010.

[41] P. L. Freddolino, S. Park, B. Roux, and K. Schulten, "Force field bias in protein folding simulations," *Biophysical*, vol. 96(9), pp. 3772–3780, 2009.

[42] L. G. Trabuco, E. Villa, E. Schreiner, C. B. Harrison, and K. Schulten, "Molecular dynamics flexible fitting: A practical guide to combine cryo-electron microscopy and x-ray crystallography," *Methods*, vol. 49(2), pp. 174–180, 2009.

[43] L. G. Trabuco, E. Villa, K. Mitra, J. Frank, and K. Schulten, "Flexible fitting of atomic structures into electron microscopy maps using molecular dynamics," *Structure*, vol. 16(5), pp. 673–683, 2008.

[44] R. McGreevy, A. Singharoy, Q. Li, J. Zhang, D. Xu, E. Perozo, and K. Schulten, "xmdff: molecular dynamics flexible fitting of low-resolution x-ray structures," *Acta Crystallographica Section D*, vol. 70, pp. 2344–2355, 2014.

[45] T. I. Croll, B. J. Smith, M. B. Margetts, J. Whittaker, M. A. Weiss, C. W. Ward, and M. C. Lawrence, "Higher-resolution structure of the human insulin receptor ectodomain: Multi-modal inclusion of the insert domain," *Structure*, vol. 24(3), pp. 469–476, 2016.

[46] T. I. Croll and G. R. Andersen, "Re-evaluation of low-resolution crystal structures via interactive molecular-dynamics flexible fitting (imdff): a case study in complement c4," *Acta Crystallographica*, vol. D72, pp. 1006–1016, 2016.

[47] J. Fennen, A. E. Torda, and W. F. van Gunsteren, "Structure refinement with molecular dynamics and a boltzmann-weighted ensemble," *Journal of Biomolecular NMR*, vol. 6(2), pp. 163–170, 1995.

[48] P. Robustelli, K. Kohlhoff, A. Cavalli, and M. Vendruscolo, "Using NMR chemical shifts as structural restraints in molecular dynamics simulations of proteins," *Structure*, vol. 18(8), pp. 923–933, 2010.

[49] B. Hess and R. M. Scheek, "Orientation restraints in molecular dynamics simulations using time and ensemble averaging," *Journal of Magnetic Resonance*, vol. 164(1), pp. 19–27, 2003.

[50] M. R. Hermann and J. S. Hub, "Saxs-restrained ensemble simulations of intrinsically disordered proteins with commitment to the principle of maximum entropy," *Journal of Chemical Theory and Computation*, vol. 15(9), pp. 5103–5115, 2019.

[51] P. Chen and J. S. Hub, "Interpretation of solution x-ray scattering by explicit-solvent molecular dynamics," *Biophysical Journal*, vol. 108(10), pp. 2573–2584, 2015.

[52] G. Fiorin, M. L. Klien, and J. Hénin, "Using collective variables to drive molecular dynamics simulations," *Molecular Physics*, vol. 111, pp. 3345–3362, 2013.

[53] W. Gronwald, S. Moussa, R. Elsner, A. Jung, B. Ganslmeier, J. Trenner, W. Kremer, K.-P. Neidig, and H. R. Kalbitzer, "Automated assignment of noesy nmr spectra using a knowledge based method (knownoe)," *Journal of Biomolecular NMR*, vol. 23(4), pp. 271–287, 2002.

[54] L.-H. Hung and R. Samudrala, "An automated assignment-free bayesian approach for accurately identifying proton contacts from noesy data," *Journal of Biomolecular NMR*, vol. 36(3), pp. 189–198, 2006.

[55] A. Cavalli, X. Salvatella, C. M. Dobson, and M. Vendruscolo, "Protein structure determination from nmr chemical shifts," *PNAS*, vol. 104(23), pp. 9615–9620, 2007.

[56] Y. Shen, O. Lange, F. Delaglio, P. Rossi, J. M. Aramini, G. Liu, A. Eletsky, Y. Wu, K. K. Singarapu, A. Lemak, A. Ignatchenko, C. H. Arrowsmith, T. Szyperski, G. T. Montelione, D. Baker, and A. Bax, "Consistent blind protein structure generation from nmr chemical shift data," *PNAS*, vol. 105(12), pp. 4685–4690, 2008.

[57] D. S. Wishart, D. Arndt, M. Berjanskii, P. Tang, J. Zhou, and G. Lin, "Cs23d: a web server for rapid protein structure generation using nmr

chemical shifts and sequence data," *Nuclear Acids Research*, vol. 36, pp. W496–W502, 2008.

[58] G. F. Schröder, A. T. Brunger, and M. Levitt, "Combining efficient conformational sampling with a deformable elastic network model facilitates structure refinement at low resolution," *Structure*, vol. 15(12), pp. 1630–1641, 2007.

[59] M. Topf, K. Lasker, B. Webb, H. Wolfson, W. Chiu, and A. Sali, "Protein structure fitting and refinement guided by cryo-em density," *Structure*, vol. 16(2), pp. 295–307, 2008.

[60] F. DiMaio, Y. Song, X. Li, M. J. Brunner, C. Xu, V. Conticello, E. Egelman, T. C. Marlovits, Y. Cheng, and D. Baker, "Atomic-accuracy models from 4.5-å cryo-electron microscopy data with density-guided iterative local refinement," *Nature Methods*, vol. 12, pp. 361–365, 2015.

[61] F. DiMaio, M. D. Tyka, M. L. Baker, W. Chiu, and DavidBaker, "Refinement of protein structures into low-resolution density maps using rosetta," *Journal of Molecular Biology*, vol. 392(1), pp. 181–190, 2009.

[62] C. C. Jolley, S. A. Wells, P. Fromme, and M. F. Thorpe, "Fitting low-resolution cryo-em maps of proteins using constrained geometric simulations," *Biophysical Journal*, vol. 94(5), pp. 1613–1621, 2008.

[63] M. Faini, F. Stengel, and R. Aebersold, "The evolving contribution of mass spectrometry to integrative structural biology," *Journal of The American Society for Mass Spectrometry*, vol. 27(6), pp. 966–974, 2016.

[64] A. M. Wollacott, L. N. Robinson, B. Ramakrishnan, H. Tissire, K. Viswanathan, Z. Shriver, and G. J. Babcock, "Structural prediction of antibody-APRIL complexes by computational docking constrained by antigen saturation mutagenesis library data," *Journal of Molecular Recognition*, vol. 32, p. e2778, 2019.

[65] D. Russel, K. Lasker, B. Webb, J. Velázquez-Muriel, E. Tjioe, D. Schneidman-Duhovny, B. Peterson, and A. Sali, "Putting the pieces together: Integrative modeling platform software for structure determination of macromolecular assemblies," *PLOS Biology*, vol. 10(1), p. e1001244, 2012.

[66] E. Karaca and A. M. J. J. J. Bonvin, "Advances in integrative modeling of biomolecular complexes," *Methods*, vol. 59(3), pp. 372–381, 2013.

[67] F. Alber, S. Dokudovskaya, L. M. Veenhoff, W. Zhang, J. Kipper, D. Devos, A. Suprapto, O. Karni-Schmidt, R. Williams, B. T. Chait, M. P. Rout, and A. Sali, "Determining the architectures of macromolecular assemblies," *nature*, vol. 450, pp. 683–694, 2007.

[68] J. Roel-Touris, C. G. Don, R. V. Honorato, J. P. G. L. M. Rodrigues, and A. J. J. J. Bonvin, "Less is more: Coarse-grained integrative modeling of large biomolecular assemblies with HADDOCK," *Journal of Chemical Theory and Computation*, vol. 15(11), pp. 6358–6367, 2019.

[69] K. Lasker, F. Förster, S. Bohn, T. Walzthoeni, E. Villa, P. Unverdorben, F. Beck, R. Aebersold, A. Sali, and W. Baumeister, "Molecular architecture of the 26s proteasome holocomplex determined by an integrative approach," *PNAS*, vol. 109(5), pp. 1380–1387, 2012.

[70] G. F. Schroöder, "Hybrid methods for macromolecular structure determination: experiment with expectations," *Current Opinion in Structural Biology*, vol. 31, pp. 20–27, 2015.

[71] F. Alber, F. Förster, D. Korkin, M. Topf, and A. Sali, "Integrating diverse data for structure determination of macromolecular assemblies," *Annual Review of Biochemistry*, vol. 77, pp. 443–477, 2008.

[72] N. Soni and M. S. Madhusudhan, "Computational modeling of protein assemblies," *Current Opinion in Structural Biology*, vol. 44, pp. 179–189, 2017.

[73] X.-L. Zhou and S.-H. Chen, "Theoretical foundation of x-ray and neutron reflectometry," *Physics Reports*, vol. 257, pp. 223–348, 1995.

[74] R. Cubitt and G. Fragneto, "Neutron reflection:: Principles and examples of applications," in *Scattering and Inverse Scattering in Pure and Applied Science*, ch. 2.8.3, pp. 1198–1208, Academic Press, 2002.

[75] D. J. McGillivray, G. Valincius, D. J. Vanderah, W. Febo-Ayala, J. T. Woodward, F. Heinrich, J. J. Kasianowicz, and M. Lösche, "Molecular-scale structural and functional characterization of sparsely tethered bilayer lipid membranes," *Biointerphases*, vol. 2, pp. 21–33, 2007.

[76] R. Budvytyte, G. Valincius, G. Niaura, V. Voiciuk, M. Mickevicius, H. Chapman, H.-Z. Goh, P. Shekhar, F. Heinrich, S. Shenoy, M. Lösche, and D. J. Vanderah, "Structure and properties of tethered bilayer lipid membranes with unsaturated anchor molecules," *Langmuir*, vol. 29(27), pp. 8645–8656, 2013.

[77] M. Lösche, J. Schmitt, G. Decher, W. G. Bouwman, and K. Kjær, "Detailed structure of molecularly thin polyelectrolyte multilayer films on solid substrates as revealed by neutron reflectometry," *Macromolecules*, vol. 31(25), pp. 8893–8906, 1998.

[78] M. Lösche, M. Piepenstock, A. Diedrich, T. Gruünewald, K. Kjær, and D. Vakin, "Influence of surface chemistry on the structural organization of monomolecular protein layers adsorbed to functionalized aqueous interfaces," *Biophysical Journal*, vol. 65, pp. 2160–2177, 1993.

[79] J. Als-Nielsen and K. Kjær, "X-ray reflectivity and diffraction studies of liquid surfaces and surfactant monolayers.," in *Phase Transitions in Soft Condensed Matter. NATO ASI Series (Series B: Physics)* (T. Riste and D. Sherrington, eds.), vol. 211, pp. 113–138, Boston, MA: Springer, 1989.

[80] C. A. Helm, H. Möhwald, K. Kjær, and J. Als-Nielsen, "Phospholipid monolayer density distribution perpendicular to the water surface. a synchrotron x-ray reflectivity study," *Europhysics Letters*, vol. 4(6), pp. 697–703, 1987.

[81] E. B. Watkins, R. J. El-khouri, C. E. Miller, B. G. Seaby, J. Majewski, C. M. Marques, and T. L. Kuhl, "Structure and thermodynamics of lipid bilayers on polyethylene glycol cushions: Fact and fiction of peg cushioned membranes," *Langmuir*, vol. 27(22), pp. 13618–13628, 2011.

[82] M. S. Kent, J. K. Murton, D. Y. Sasaki, S. Satija, B. Akgun, H. Nanda, J. E. Curtis, J. Majewski, C. R. Morgan, and J. R. Engen, "Neutron reflectometry study of the conformation of hiv nef bound to lipid membranes," *Biophysical Journal*, vol. 99(6), pp. 1940–1948, 2010.

[83] E. Y. Chi, C. Ege, A. Winans, J. Majewski, G. Wu, K. Kjær, and K. Y. C. Lee, "Lipid membrane templates the ordering and induces

the fibrillogenesis of alzheimer's disease amyloid-β peptide," *Proteins*, vol. 72, pp. 1–24, 2008.

[84] M. S. Kent, H. Kim, J. K. Murton, S. Satija, J. Majewski, and I. Kuzmenko, "Oligomerization of membrane-bound diphtheria toxin (crm197) facilitates a transition to the open form and deep insertion," *Biophysical Journal*, vol. 94(6), pp. 2115–2127, 2008.

[85] S. Shenoy, R. Moldovan, J. Fitzpatrick, D. J. Vanderah, M. Deserno, and M. Löesche, "In-plane homogeneity and lipid dynamics in tethered bilayer lipid membranes (tBLMs)," *Soft Matter*, vol. 6, pp. 1263–1274, 2010.

[86] J. F. Ankner and C. Majkrzak, "Subsurface profile refinement for neutron specular reflectivity (invited paper)," *Proceedings SPIE, Neutron Optical Devices and Applications*, vol. 1738, pp. 260–269, 1992.

[87] P. Shekhar, H. Nanda, M. Lösche, and F. Heinrich, "Continuous distribution model for the investigation of complex molecular architectures near interfaces with scattering techniques," *Journal of Applied Physics*, vol. 110, pp. 102216–1–102216–12, 2011.

[88] S. Shenoy, P. Shekhar, F. Heinrich, M. C. Daou, A. Gerick, A. H. Ross, and M. Lösche, "Membrane association of the PTEN tumor suppressor: Molecular details of the protein-membrane complex from SPR binding studies and neutron reflection," *PLoS ONE*, vol. 7, p. e32591, 2012.

[89] B. Roux and J. Weare, "On the statistical equivalence of restrained-ensemble simulations with the maximum entropy method," *The Journal Of Chemical Physics*, vol. 138, p. 084107, 2013.

[90] P. F. Flynn, R. J. B. Urbauer, H. Zhang, A. L. Lee, and J. A. Wand, "Main chain and side chain dynamics of a heme protein: ^{15}N and ^{2}H NMR relaxation studies of R.capsulatus Ferrocytochrome c_2," *Biochemistry*, vol. 40(22), pp. 6559–6569, 2001.

[91] Y. He, J. Y. Chen, J. R. Knab, W. Zheng, and A. G. Markelz, "Evidence of protein collective motions on the picosecond timescale," *Biophysical Journal*, vol. 100(4), pp. 1058–1065, 2011.

[92] H. Nanda, F. Heinrich, and M. Lösche, "Membrane association of the PTEN tumor suppressor: Neutron scattering and MD simulations reveal the structure of protein-membrane complexes," *Methods*, vol. 77-78, pp. 136–146, 2015.

[93] S. Shenoy, H. Nanda, and M. Lösche, "Membrane association of the PTEN tumor suppressor: Electrostatic interaction with phosphatidylserine-containing bilayers and regulatory role of the C-terminal tail.," *Journal of Structural Biology*, vol. 180, pp. 394–408, 2012.

[94] M. P. Myers, J. P. Stolarov, C. Eng, J. Li, S. I. Wang, M. H. Wigler, R. Parsons, and N. K. Tonks, "P-TEN, the tumor suppressor from human chromosome 10q23, is a dual-specificity phosphatase," *Proceeding of the National Academy of Science USA*, vol. 94, pp. 9052–9057, 1997.

[95] V. Stambolic, A. Suzuki, J. de la Pompa, G. Brothers, C. Mirtsos, T. Sasaki, J. Ruland, J. Penninger, D. Siderovski, and T. Mak, "Negative regulation of PKB/Akt-dependent cell survival by the tumor suppressor PTEN," *Cell*, vol. 95, pp. 29–39, 95.

[96] T. Maehama and E. J. Dixon, "The tumor suppressor, PTEN/MMAC1, dephosphorylates the lipid second messenger, phosphatidylinositol 3,4,5-trisphosphate," *Journal of Biological Chemistry*, vol. 273, pp. 13375–13378, 1998.

[97] L. Simpson and R. Parsons, "PTEN: Life as a tumor suppressor," *Experimental Cell Research*, vol. 264, pp. 29–41, 2001.

[98] J. O. Lee, H. Yang, M. M. Georgescu, A. Di Cristofano, T. Maehama, Y. Shi, J. E. Dixon, P. P. Pandolfi, and N. P. Pavletich, "Crystal structure of the PTEN tumor suppressor: Implications for its phosphoinositide phosphatase activity and membrane association," *Cell*, vol. 99, pp. 323–334, 1999.

[99] M. Rahdar, T. Inoue, T. Meyer, J. Zhang, F. Vazquez, and P. N. Devreotes, "A phosphorylation-dependent intramolecular interaction regulates the membrane association and activity of the tumor suppressor PTEN," *Proceedings of the National Academy of Science USA*, vol. 106, pp. 480–485, 2009.

[100] A. H. Ross and A. Gericke, "Phosphorylation keeps PTEN phosphatase closed for business," *Proceeding of the National Academy of Science USA*, vol. 106, pp. 1297–1298, 2009.

[101] D. T. Gillespie, "A theorem for physicists in the theory of random variables," *American Journal of Physics*, vol. 51, pp. 520–532, 1983.

[102] L. Li and A. H. Ross, "Why is pten an important tumor suppressor," *Journal of Cellular Biochemistry*, vol. 102, no. 6, pp. 1368–1374, 2007.

[103] C. Lanczos, *The Variational Principles Of Mechanics*. Mineola, New York: Dover Publications, Inc., 1986.

[104] F. Heinrich and M. Lösche, "Zooming in on disordered systems: Neutron reflection studies of proteins associated with fluid membranes," *Biochimica et Biophysica Acta*, vol. 1838(9), pp. 2341–2349, 2014.