

DISSERTATION

IMPLICIT SOLVATION USING THE SUPERPOSITION APPROXIMATION APPLIED TO  
MANY-ATOM SOLVENTS WITH STATIC GEOMETRY AND ELECTROSTATIC DIPOLE

Submitted by

Max Atticus Mattson

Department of Chemistry

In partial fulfillment of the requirements

For the Degree of Doctor of Philosophy

Colorado State University

Fort Collins, Colorado

Spring 2020

Doctoral Committee:

Advisor: Amber T. Krummel

Co-Advisor: Martin McCullagh

Grzegorz Szamel

Amy Prieto

David Krueger

Copyright by Max A. Mattson 2020

All Rights Reserved

## ABSTRACT

### IMPLICIT SOLVATION USING THE SUPERPOSITION APPROXIMATION APPLIED TO MANY-ATOM SOLVENTS WITH STATIC GEOMETRY AND ELECTROSTATIC DIPOLE

Large-scale molecular aggregation of organic molecules, such as perylene diimides, is a phenomenon that continues to generate interest in the field of solar light-harvesting. Functionalization of the molecules can lead to different aggregate structures which in turn alter the spectroscopic properties of the molecules. To improve the next generation of perylene diimide solar cells a detailed understanding of their aggregation is necessary. A critical aid in understanding the spectroscopic properties of large-scale aggregating systems is molecular simulation. Thus development of an efficient and accurate method for simulating large-scale aggregating systems at dilute concentrations is imperative.

The Implicit Solvation Using the Superposition Approximation model (IS-SPA) was originally developed to efficiently model nonpolar solvent–solute interactions for chargeless solutes in TIP3P water, improving the efficiency of dilute molecular simulations by two orders of magnitude. In the work presented here, IS-SPA is developed for charged solutes in chloroform solvent. Chloroform is the first solvent model developed for IS-SPA that is composed of more than one Lennard-Jones potential. Solvent distribution and force histograms were measured from all-atom explicit-solvent molecular dynamics simulations, instead of using analytic functions, and tested for Lennard-Jones sphere solutes of various sizes. The level of detail employed in describing the 3-dimensional structure of chloroform is tested by approximating chloroform as an ellipsoid, spheroid, and sphere by using 3-, 2-, and 1-dimensional distribution and force histograms respectively.

A perylene diimide derivative, lumogen orange, was studied for its unfamiliar aggregation mechanism in chloroform and tetrahydrofuran solvents via Fourier-transform infrared and 2-dimensional infrared spectroscopies as well as all-atom explicit-solvent molecular dynamics simulations and quantum mechanical frequency calculations. Molecular simulations identified two categories of likely aggregate dimer structures: the expected  $\pi$ -stack structure, and a less familiar edge-sharing structure where the most highly charged atoms of the perylene diimide core are

strongly interacting. Quantum mechanical vibrational frequency calculations were performed for various likely dimer aggregate structures identified in molecular simulation and compared to experimental spectroscopic results. The experimental spectra of the aggregating system share qualities with the edge-sharing dimer frequency calculations however larger aggregate structures should be tested.

A violanthrone derivative, violanthrone-79 (V-79), was studied for its differing aggregation mechanisms in chloroform and tetrahydrofuran solvents via Fourier-transform infrared and 2-dimensional infrared spectroscopies as well as all-atom explicit-solvent molecular dynamics simulations and quantum mechanical frequency calculations. The  $\pi$ -stacking aggregate structure of V-79 is supported by all methods used, however, the type of  $\pi$ -stacking orientations are different between the two solvents. Chloroform supports parallel  $\pi$ -stacked aggregates while tetrahydrofuran supports anti-parallel  $\pi$ -stacked aggregates which show differing vibrational energy delocalization between the aggregated molecules.

The publications in chapters 3 and 4 demonstrate the power of combining experimental spectroscopy and computational methods like molecular dynamics simulations and quantum mechanical frequency calculations, however, they also show how having larger simulations with multiple solute molecules are needed. This is why developing IS-SPA to be used for these simulations is necessary. Further developments to IS-SPA are discussed regarding the importance of various symmetries of chloroform and the subsequent dimensionalities of the histograms used to describe its distribution and Lennard-Jones force. Two methods for describing the Coulombic forces of chloroform solvation are discussed and tested on oppositely charged Lennard-Jones sphere solutes. The radially symmetric treatment fails to capture the Coulombic forces of the spherical solute system from all-atom explicit-solvent molecular dynamics simulations. A dipole polarization treatment is presented and tested for the charged spherical solute system which better captures the Coulombic forces measured from all-atom explicit-solvent molecular dynamics simulations.

Additional considerations for the improvement of IS-SPA and the developments in this work are presented. The dipole polarization approximation outlined in chapter 5 assumes that each chloroform is a static dipole, allowing the dipole magnitude to fluctuate as well as polarize is a more physically rigorous approximation that will likely improve the accuracy of Coulombic

forces in IS-SPA. A novel method, drawn from the knowledge gained studying chloroform, for the efficient modeling of new solvent types including flexible solvent molecules in IS-SPA is discussed.

## ACKNOWLEDGEMENTS

To Dr. Amber Krummel, thank you for your support and guidance, it has been an honor working for and with you. To Dr. Brad Luther, thank you for your friendship and a different sort of guidance. To Dr. Martin McCullagh, thank you for welcoming me into your group, working with me, and teaching me. To Dr. Rex Lake, thank you for all that you have taught me and I am sure still will teach me. Thank you all, I could not have done this without you. Lastly, thank you to my friends and family, you have given me more than I can ever hope to repay.

## TABLE OF CONTENTS

	ABSTRACT . . . . .	ii
	ACKNOWLEDGEMENTS . . . . .	v
Chapter 1	Introduction . . . . .	1
Chapter 2	Methods . . . . .	6
2.1	All-Atom Molecular Dynamics . . . . .	6
2.1.1	Replica-Exchange Umbrella Sampling . . . . .	6
2.2	IS-SPA Theory . . . . .	8
2.2.1	Superposition Approximation . . . . .	9
2.2.2	Histogram Atomic Distribution and Force Functions . . . . .	10
2.2.2.1	Dimensionality of Histograms . . . . .	10
2.2.2.2	3-Dimensional Histograms: Ellipsoidally Symmetric Solvent Approximation . . . . .	11
2.2.2.3	2-Dimensional Histograms: Spheroidally Symmetric Solvent Approximation . . . . .	14
2.2.2.4	1-Dimensional Histograms: Spherically Symmetric Solvent Approximation . . . . .	15
2.2.2.5	Cubic Spline Interpolation . . . . .	16
2.2.2.6	Analytic Extrapolation of Histograms . . . . .	18
2.2.3	Model System . . . . .	19
2.2.4	Measuring Molecular Dynamics Simulation . . . . .	21
2.2.4.1	Model System Measurements . . . . .	21
2.2.4.2	Chemical System Measurements . . . . .	26
2.2.5	Fitting Histograms . . . . .	27
2.2.5.1	Bayesian Inference . . . . .	29
2.2.5.2	Poisson Regression . . . . .	32
Chapter 3	Elucidating Structural Evolution of Perylene Diimide Aggregates Using Vibrational Spectroscopy and Molecular Dynamics Simulations . . . . .	35
3.1	Overview . . . . .	35
3.2	Introduction . . . . .	36
3.3	Experimental Methods . . . . .	38
3.3.1	Sample Preparation . . . . .	38
3.3.2	Linear IR and 2D IR Spectroscopy . . . . .	38
3.4	Computational Methods . . . . .	39
3.4.1	Molecular Dynamics . . . . .	39
3.4.2	Quantum Calculations . . . . .	40
3.5	Results and Discussion . . . . .	41
3.6	Conclusion . . . . .	52
3.7	Supporting Information . . . . .	52
3.7.1	Computational Methods . . . . .	52
3.7.1.1	Molecular Dynamics . . . . .	52

Chapter 4	Vibrational Properties of Solvent Dependent Violanthrone-79 Aggregates Using Two-Dimensional Infrared Spectroscopy and Molecular Dynamics Simulations . . . . .	55
4.1	Overview . . . . .	55
4.2	Introduction . . . . .	55
4.3	Experimental and Computational Methods . . . . .	58
4.3.1	Sample Preparation . . . . .	58
4.3.2	2DIR Spectroscopy . . . . .	58
4.3.3	Molecular Dynamics Simulations . . . . .	59
4.3.4	Quantum Calculations . . . . .	60
4.4	Results and Discussion . . . . .	60
4.5	Conclusion . . . . .	66
4.6	Supporting Information . . . . .	67
Chapter 5	Development of IS-SPA for Chloroform . . . . .	69
5.1	Non-Spherical Solvent . . . . .	69
5.2	1-Dimensional Histogram Considerations . . . . .	72
5.3	Dipole Coulombic Forces in IS-SPA . . . . .	77
5.3.1	Polarization Theory . . . . .	77
5.3.2	Long Range Behavior . . . . .	80
5.3.3	Application to Model System . . . . .	82
Chapter 6	Future Work . . . . .	87
6.1	Dynamic Dipole in IS-SPA . . . . .	87
6.2	IS-SPA Applied to Flexible Solvent Molecules . . . . .	89
6.2.1	Methyl-Atom Strategy . . . . .	90
Appendices	. . . . .	100
1	Simulation Preparation . . . . .	101
1.1	Making a Molecule in Gaussian . . . . .	101
1.1.1	Frequency Calculation . . . . .	102
1.1.2	Generate Electro-Static Potential from Optimized Geometry . . . . .	102
1.1.3	Restraints . . . . .	105
2	Code . . . . .	105
2.1	Model System IS-SPA Histogram Measurement Code: <code>gr2d.p2.py</code> . . . . .	105
2.2	Chemical System IS-SPA 3D Histogram Measurement Code: <code>gr3d.p2.py</code> . . . . .	115
2.3	Fitting IS-SPA 3D Histogram $\chi^2$ : <code>SPA.poisson.omp.rc16.sym.f</code> . . . . .	121
2.4	Poisson Regression Fitting Code: <code>SPA.poisson2.omp.rc16.sym.f</code> . . . . .	127
2.5	IS-SPA 1-Dimensional: <code>isspa1.1D.f90</code> . . . . .	133
2.6	IS-SPA 2-Dimensional: <code>isspa1.2D.f90</code> . . . . .	152
2.7	IS-SPA 3-Dimensional: <code>isspa1.3D.f90</code> . . . . .	169
2.8	IS-SPA 1-Dimensional with Dipole CDD: <code>isspa2.1D.f90</code> . . . . .	190
2.9	Fortran Function Subroutines: <code>functions.f90</code> . . . . .	211



# Chapter 1

## Introduction

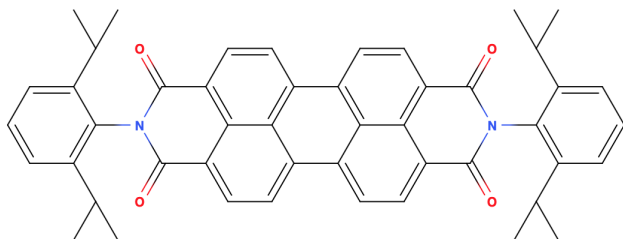
The goal of my graduate research at Colorado State University was to develop an implicit solvation model for chloroform to be used in molecular simulations probing large-scale aggregation in systems like the perylene diimide (PDI) and violanthrone systems my research group was studying spectroscopically. In our spectroscopic experiments it was a common theme to observe an interesting shift or change in the spectrum, likely due to aggregation, but require quantum mechanical (QM) calculations to identify what vibrational modes of the molecule were involved. This process of comparing experimental vibrational spectra to the QM vibrational frequency calculations of the probe molecule worked well for systems where the method of aggregation was already known or easily guessed. However, for systems where the method of aggregation was less constrained or understood, the parameter space of likely aggregation structures and thus spectral changes due to mode coupling was far too large to explore manually. For this reason we started using molecular dynamics (MD) simulations to understand what structures were likely responsible for the spectroscopic features we were observing experimentally. Combining MD simulations with QM frequency calculations of the most likely small aggregate structures found in MD proved to be a useful technique in our two publications presented in chapters 3 and 4. However, if the aggregation mechanism was more complex perhaps requiring many more than two solute molecules to form, the simulation of a much larger aggregate system was prohibitively expensive using standard all-atom methods of MD. This necessitated the development of a method to increase the efficiency of MD simulations without sacrificing the accuracy of all-atom MD simulations.

In my project, we have used all-atom explicit-solvent molecular dynamics (AESMD) simulations, where every atom is modeled and propagated explicitly to give high-accuracy results with regard to the intermolecular structures found in aggregates. However, AESMD simulations are computationally expensive because every solute and solvent atom requires a force for the propagation of the system; as system size grows the computational time required for sampling becomes intractable. The vast number of solvent atoms in these systems in many cases far outnumber the solute atoms and thus take the majority of the computation time. Compounding the problem is the fact that

many of the solvent atoms are located in the bulk of the solvent where their behavior is typically uninteresting.

For an MD simulation to be physically relevant the solvent behavior must be accurate, however the computational time spent in the bulk of the solvent is in some sense wasted as the forces calculated have little to no direct effect on the solute being studied. For instance, for a protein simulation in water, the water needs to behave accurately when near the protein and impart appropriate forces but for the water molecules that are 20 or 30 Å away their behavior is either unimportant in reference to Lennard-Jones (LJ) forces or is uninteresting and somewhat generalizable for long-range Coulombic forces. For aggregating systems where the simulation is run at a specific concentration to mimic experimental studies, the solutes might start the simulation dispersed throughout the box with a large solvent-accessible surface area, however as they aggregate some of that surface area gets replaced with other solutes. Eventually when the solutes of the system are aggregating together there ends up being a vast volume of the simulation box where there are only solvents which is again essentially wasted computational time.

To effectively simulate large aggregating systems we need a method that is both efficient and accurate. AESMD simulations are accurate at simulating molecular systems but lack the efficiency to do so effectively as those systems increase in size. Other implicit solvation models either lack the accuracy, such as Poisson–Boltzmann<sup>1</sup> and generalized Born<sup>2</sup> models, or lack the efficiency, such as reference interaction site models,<sup>3,4</sup> to make them viable solutions to this problem.



**Figure 1.1:** Chemical structure of Lumogen Orange a PDI derivative.

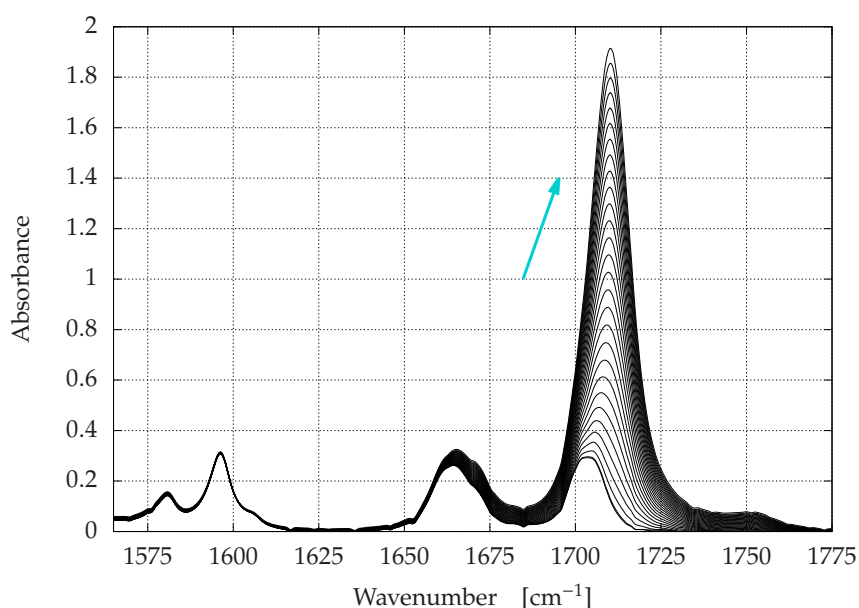
<sup>1</sup>Baker, N. A. *Current Opinion in Structural Biology* **2005**, *15*, 137–143.

<sup>2</sup>Onufriev, A.; Bashford, D.; Case, D. A. *The Journal of Physical Chemistry B* **2000**, *104*, Publisher: American Chemical Society, 3712–3720.

<sup>3</sup>Chandler, D.; Andersen, H. C. *The Journal of Chemical Physics* **1972**, *57*, 1930–1937.

<sup>4</sup>Beglov, D.; Roux, B. *The Journal of Chemical Physics* **1996**, *104*, 8678–8689.

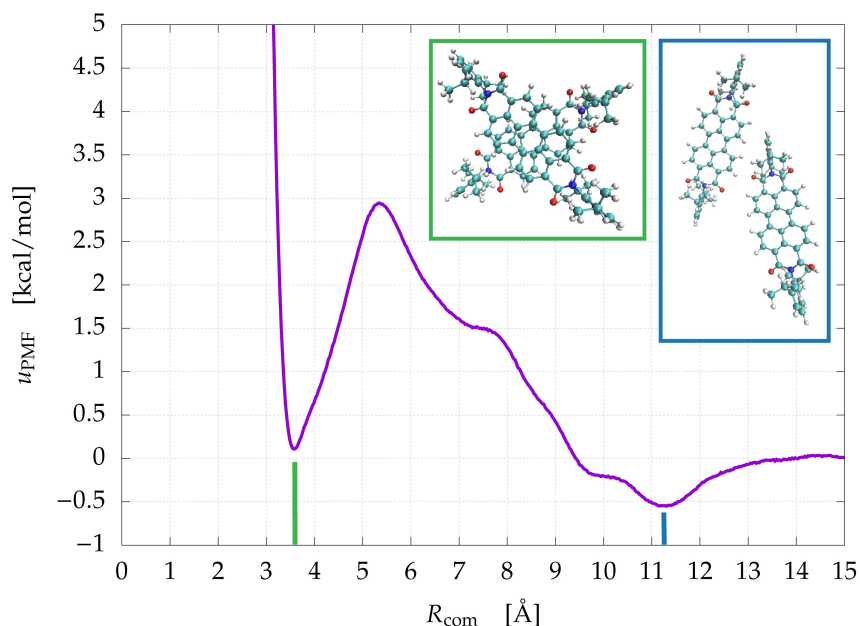
The importance of developing more efficient simulation methods can be seen by looking at the first two manuscripts from my work, chapter 3 and 4. In these studies we measured an experimental observable like the one shown in figure 1.2, where the vibrational spectrum of Lumogen Orange (LO)—a PDI derivative (figure 1.1)—is evolving over the course of hours. From an experimental perspective it is difficult to say what molecular-level details are leading to this event apart from some kind of general aggregation. To understand what the molecular structure of this aggregation is we would need to have structures to use for quantum mechanical frequency calculations. So we must have a way to generate likely structures of our aggregating system.



**Figure 1.2:** FTIR spectra of LO in chloroform at 5 mM concentration. The time separation between each spectrum is 15 minutes. The two lower frequency modes ( $1585\text{ cm}^{-1}$  and  $1595\text{ cm}^{-1}$ ) are ring vibrations of the perylene core that do not grow in as a function of time. The two higher frequency modes ( $1665\text{ cm}^{-1}$  and  $1705\text{ cm}^{-1}$ ) are carbonyl vibrations that both blue-shift and grow in intensity as a function of time, especially the high energy carbonyl mode.

One way to identify likely structures of aggregation is measure a potential of mean force (PMF) of the simplest aggregate system, a LO dimer, using AESMD simulations. The PMF, figure 1.3, which shows free energy as a function of center-of-mass separation distance between the solutes, allows us to see the most likely structures of our dimer aggregate. In the case of LO we found two minima: the expected  $\pi$ -stacked structure for a PDI, and an unexpected edge-on structure where the negatively charged oxygens of the imide-group interact with the positively charged bay-hydrogens of the

perylene core.<sup>5</sup> Generating this PMF was important for this work because it identified several structures to calculate vibrational spectra from that ended up enhancing our understanding of the chemistry we were observing spectroscopically. The problem with our approach was that it could not be scaled up to a system size more representative of the experimental system because our simulations were performed with AESMD.



**Figure 1.3:** PMF of LO in AESMD as a function of center-of-mass displacement between the two LO molecules. Representative dimer structures for each well are shown in color-coded inset images that correspond to the  $\pi$ -stacked and edge-on dimer structures respectively.

AESMD simulations are useful and important tools for accurately simulating the systems they model but they are too expensive to simulate a large aggregate system like ours especially at experimental solute concentrations. In AESMD the same amount of time is being spent on each solvent atom as each solute atom. At the experimental concentration used in our first study,<sup>5</sup> 5 mM LO in chloroform, there are roughly 135 chloroform atoms per solute atom so the solvent is taking the vast majority of the computational time in our simulation. To approach a more physical system size of tens or hundreds of LO solute molecules where there are enough solutes to form

<sup>5</sup>Mattson, M. A. et al. *The Journal of Physical Chemistry B* **2018**, *122*, 4891–4900.

more complex multi-molecular aggregates we will need a tool that allows us to model the solvation of our system accurately but without the massive amount of time spent on the solvent.

The Implicit Solvation Using the Superposition Approximation (IS-SPA) model seeks to accomplish this very thing for TIP3P water.<sup>6</sup> IS-SPA works by representing the solvent as an analytic function for both the distribution and Lennard-Jones (LJ) force functions and using Monte-Carlo integration in simulation to approximate the mean solvent force for each solute atom. IS-SPA was developed only for chargeless solutes, so to expand this tool to be used for our systems, PDI and violanthrone solutes in chloroform solvent, we will need to consider how the LJ force can be treated for a multi-atom solvent and how the Coulombic force will be treated in the context of IS-SPA.

In chapter 2, I outline the background theory of IS-SPA and cover some details of how the method was implemented in my work. Chapters 3 and 4 are manuscripts focused on LO and violanthrone-79 chemical systems respectively with equal parts experimental spectroscopy, molecular simulation and frequency calculation. The manuscripts highlight the need for a method to run larger simulations of these aggregating systems. Chapter 5 contains developments in IS-SPA that have been made for implementing chloroform solvent at different levels of complexity for both LJ and Coulombic forces. Chapter 6 poses some of my ideas for further improving IS-SPA for existing solvent systems as well as developing IS-SPA for fundamentally different solvents. Finally, appendices 1 and 2 contain a guide for setting up molecular simulations in AMBER<sup>7</sup> and links to the codes written and used for the work presented in this dissertation.

---

<sup>6</sup>Lake, P. T.; McCullagh, M. *Journal of Chemical Theory and Computation* **2017**.

<sup>7</sup>Case, D. et al. AMBER16 Package.

# Chapter 2

## Methods

### 2.1 All-Atom Molecular Dynamics

All-atom explicit-solvent molecular dynamics (AESMD) simulations calculate the forces on every solute and solvent atom of the system explicitly for propagation. AESMD simulations are useful for the modeling of chemical systems because the explicit modeling of each atom results in a high degree of accuracy compared to other models that might coarse-grain solute or solvent atoms to attain higher efficiency. For this reason, we will use AESMD simulations as the gold standard when assessing the accuracy of our implicit solvation theory, Implicit Solvation Using the Superposition Approximation (IS-SPA).<sup>6</sup> Specifically, we will be seeking to reproduce the potential of mean force (PMF) of a dimerizing molecular system as a first step in modeling large-scale molecular aggregation.

To measure the PMF, umbrella sampling (US) can be used to enhance the sampling of less stable configurations along the collective variable. US-MD simulations enhance the sampling of barriers by applying an artificial biasing potential to the collective variable which constrains the simulation to sample configurations near the minimum of the biasing potential. In the case of our chemical system, Lumogen Orange (LO), US alone was insufficient to ergodically sample the barrier of dimerization as a function of center-of-mass separation distance between the two LO molecules so replica-exchange umbrella sampling (REUS) was used.

#### 2.1.1 Replica-Exchange Umbrella Sampling

REUS simulations are a type of US simulation where the biasing potentials get periodically swapped with neighboring potentials in an attempt to further enhance the sampling within and between each US window.<sup>8-10</sup> Sometimes in US simulations, such as our LO simulations in chloroform, the configurations sampled in the bias-window on one side of a barrier are incongruous with those sampled in the bias-window on the other side of the barrier. For instance, if a dimer system is

---

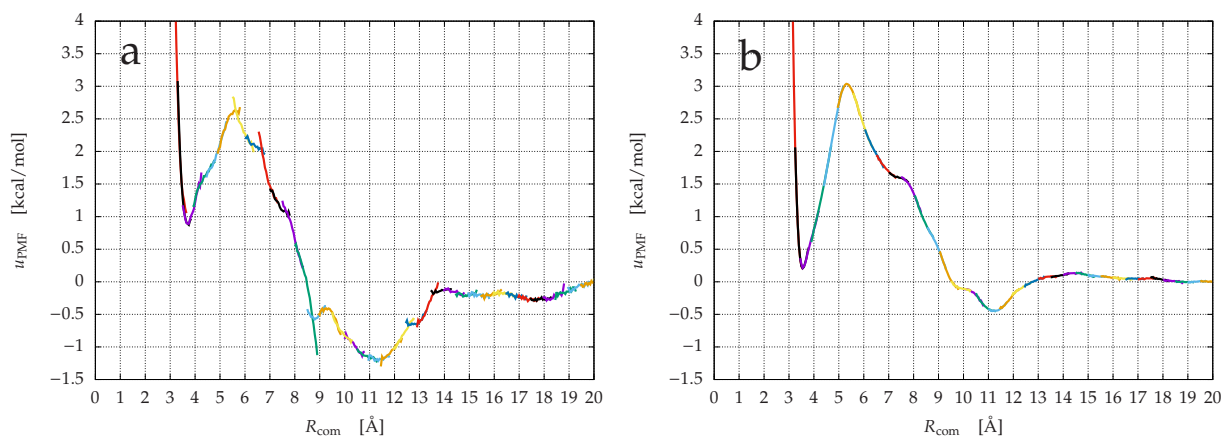
<sup>6</sup>Lake, P. T.; McCullagh, M. *Journal of Chemical Theory and Computation* **2017**.

<sup>8</sup>Sugita, Y.; Okamoto, Y. *Chemical Physics Letters* **1999**, *314*, 141–151.

<sup>9</sup>Sugita, Y.; Kitao, A.; Okamoto, Y. *The Journal of chemical physics* **2000**, *113*, 6042–6051.

<sup>10</sup>Fukunishi, H.; Watanabe, O.; Takada, S. *The Journal of Chemical Physics* **2002**, *116*, 9058–9067.

biased on the left side of the barrier it may preferentially sample certain configurations where, for example, the molecules are minimally rotated with respect to each other as the molecules separate and climb the barrier. However, when biased to the right side of the barrier it may be preferred for the molecules to come together in a rotated geometry due to the nature of the most stable states on the far-side of the barrier. In a situation like this, where the states sampled on one side of a barrier have distinctly different orientations with the states sampled on the other side of the barrier, we can sometimes get non-ergodic sampling which manifests as individual bias-window PMFs unsmoothly stitching together in the overlap region along the collective variable between the two window-bias minima as shown in figure 2.1a.



**Figure 2.1:** The PMF of dimerization of LO measured using (a) US and (b) REUS to sample the center-of-mass separation distance between the two molecules. The unsmoothly stitched windows in (a) are the result of non-ergodic sampling between those windows.

To ameliorate this situation REUS can increase the number of configurations sampled between adjacent biasing windows by increasing the diversity of starting configurations of the system as it begins the process of crossing the barrier. By diversifying the starting configurations of the dimer as it moves from its previous window to its current exchanged window, the dimer more fully samples the overlap between adjacent windows thus allowing for a smoother PMF as shown in figure 2.1b.

## 2.2 IS-SPA Theory

In AESMD simulations the solvent behavior and effect on the solute is captured by explicitly modeling every atom of the solvent. However, in solvated MD simulations where the solute behavior is the focus of study, what is important is capturing the average behavior of the solvent on the solute because the average force on solute  $i$ ,  $\langle f_i \rangle$  is related to the potential of mean force (PMF) of that solute  $i$  through the gradient,

$$\langle f_i \rangle = \nabla_i \overbrace{[T \ln G(\mathbf{R}^N)]}^{\substack{\text{free energy} \\ \text{i.e. PMF}}} \quad (2.1)$$

where  $T$  is the temperature in units of energy and  $\langle f_i \rangle$  is the mean force on the  $i^{\text{th}}$  solute atom. The right hand side of equation (2.1) is the gradient of the negative free energy with respect to the  $i^{\text{th}}$  solute which gives the mean force since  $G(\mathbf{R}^N)$  is the distribution of the solutes. The PMF is the free energy of the system along the chosen collective variable. Getting the PMF correct is important because it is the connection to the thermodynamics of the system. Getting the free energy, PMF, correct means that the Boltzmann weightings of each state are correct, which means that the connection to thermodynamics is there. Thus the goal of an implicit solvation model is to accurately predict the mean force of the solvent on the solutes without having to simulate all of the solvent degrees of freedom, thus increasing efficiency.

One of the most fundamental things a solvent does to a solute is apply a force to it. This is what the Implicit Solvation using the Superposition Approximation<sup>6</sup> (IS-SPA) model tries to replicate through implicit solvation. The mean force a molecule feels from a solvent can be represented exactly as,

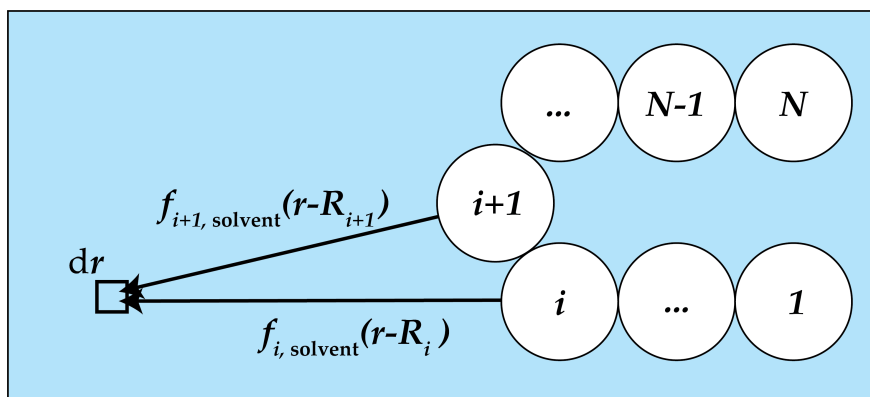
$$\langle f_i \rangle_{\text{solvent}} = \rho \int d\mathbf{r} f_{i,\text{solvent}}(\mathbf{r} - \mathbf{R}_i) g(\mathbf{r}; \mathbf{R}^N) \quad (2.2)$$

where  $\mathbf{r}$  represents the position of a solvent, and  $\mathbf{R}_i$  is the position of the  $i^{\text{th}}$  solute atom. The mean force on solute particle  $i$  from the solvent,  $\langle f_i \rangle_{\text{solvent}}$ , is equal to the force that a solvent molecule at a given distance,  $|\mathbf{r} - \mathbf{R}_i|$ , would put on solute atom  $i$ ,  $f_{i,\text{solvent}}(\mathbf{r} - \mathbf{R}_i)$ , multiplied by the relative probability with respect to bulk of a solvent molecule being at that location given by the solvent

<sup>6</sup>Lake, P. T.; McCullagh, M. *Journal of Chemical Theory and Computation* 2017.



distribution function  $g(\mathbf{r}; \mathbf{R}^N)$ , which takes into account all solute positions  $\mathbf{R}^N$  to give the solvent density at  $\mathbf{r}$ . This is all integrated for all possible solvent locations,  $d\mathbf{r}$ , and the relative probability is turned into an absolute probability with the bulk density of the solvent,  $\rho$ . An example system is shown in figure 2.2 with solvent forces acting on solute atoms from an infinitesimal solvent volume.



**Figure 2.2:** The average force of the solvent is calculated for each solute atom. Each solvent position is weighted by the molecular distribution function. Depicted are two attractive forces that the solvent at  $d\mathbf{r}$  is exerting on solute atoms  $i$  and  $i + 1$ .

Equation (2.2) has two components that describe the nature of the solvent: the direct force, and the molecular density. There are a number of approximations that can be made to simplify these components, the first we will discuss is an approximation for the molecular distribution function,  $g(\mathbf{r}; \mathbf{R}^N)$ .

### 2.2.1 Superposition Approximation

The first approximation we use is the Kirkwood Superposition Approximation<sup>11</sup> (SPA) to turn the complicated functional form of  $g(\mathbf{r}; \mathbf{R}^N)$ , which is the distribution function of solvent around the entire solute molecule, into a much simpler form by breaking the molecular form into the product of the radial distribution functions of each solute atom  $i$ ,  $g_i(|\mathbf{r} - \mathbf{R}_i|)$ .

<sup>11</sup>Kirkwood, J. G. *The Journal of Chemical Physics* **1935**, 3, 300–313.

$$\underbrace{g(\mathbf{r}; \mathbf{R}^N)}_{\text{whole molecule}} \approx \prod_{i=1}^N \overbrace{g_i(|\mathbf{r} - \mathbf{R}_i|)}^{\text{individual atoms}} \quad (2.3)$$

By employing SPA in this situation, we go from having an intractable  $3N$ -dimensional problem for the molecular distribution function to a set of  $N$  tractable atomic distribution functions and their product. The SPA performs well for non-bonded systems but fails for bonded systems. It fails in locations where the atomic distribution functions constructively interfere and you get the unphysical result of multiplicative density close to the solutes.

## 2.2.2 Histogram Atomic Distribution and Force Functions

We used histograms measured from explicit simulation data for modeling the individual atomic densities and forces. Histograms were used because chloroform has a non-spherical LJ potential unlike TIP3P water<sup>12</sup> which was used in the initial development of IS-SPA.<sup>6</sup>

Using histograms for the density and force has the advantage of being sampled directly from AESMD simulation data. The histograms aren't constrained to follow a particular functional form which makes them more flexible as well as more generally applicable than analytic forms for modeling the solvent. However, histograms are discrete so they require interpolation. There are many methods for interpolating between discrete data, the method employed in this work is cubic spline interpolation. Cubic spline interpolation is briefly discussed along with other relevant processes performed for the use of histograms later in sections 2.2.2.5 and 2.2.2.6.

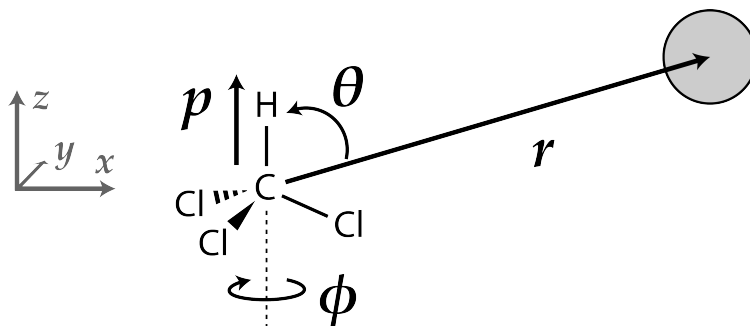
### 2.2.2.1 Dimensionality of Histograms

Chloroform has a tetrahedral geometry with  $C_{3v}$  symmetry where the axis of rotation is the polar axis of the molecule, the C-H bond. To completely model a chloroform molecule we would need  $3N = 15$  degrees of freedom to explicitly control the position of each atom. However, this is what we are trying to get away from with implicit solvation so we can make a few *hopefully* reasonable assumptions and model chloroform satisfactorily with 3 degrees of freedom or less as shown in

<sup>12</sup>Jorgensen, W. L. et al. *The Journal of Chemical Physics* **1983**, 79, 926–935.

<sup>6</sup>Lake, P. T.; McCullagh, M. *Journal of Chemical Theory and Computation* **2017**.

figure 2.3. In this section, for the sake of brevity, the magnitude of the separation vector between solute and solvent,  $|\mathbf{r} - \mathbf{R}_i|$ , will be replaced with  $r$ .



**Figure 2.3:** Depiction of the 3 intermolecular degrees of freedom of chloroform with respect to a spherical solute particle. The distance from solvent to solute where the vector  $r$  points from solvent center to solute, the polar tilt angle with respect to  $r$ ,  $\theta$ , and the azimuthal twist angle of the solvent about its polar axis,  $\phi$ . In practice the cosine of the tilt angle is used so that the Jacobian is unity. Thus  $\cos \theta = 1$  corresponds to the hydrogen pointing towards the solute and  $\cos \theta = -1$  corresponds to the hydrogen pointing away from the solute.

If we assume that chloroform has a rigid structure such that its internal coordinates can't be deformed we are left with  $3N - (3N - 6) = 6$  degrees of freedom corresponding to the three translations and three rotations of the solvent. The three translational degrees of freedom can be further simplified to a scalar distance between the solute atom and the chloroform because the solute atoms are spherically symmetric. Furthermore, the rotations can be simplified from two degenerate rotations that correspond to a polar axis tilt about  $x$  and  $y$  and one rotation about the polar axis,  $z$ ; to the tilt of the polar axis with respect to  $r$ ,  $\theta$ , and the twist about the polar axis,  $\phi$ , resulting in a total of two rotations. We will call these three variables:  $r$ ,  $\cos \theta$ , and  $\phi$ , where  $r$  is the distance from solute center to the carbon of chloroform,  $\cos \theta = \hat{r} \cdot \hat{p}$  i.e. the tilt of the chloroform dipole vector with respect to the solute, and  $\phi$  the azimuthal twist angle of the chlorines about the polar axis.

### 2.2.2.2 3-Dimensional Histograms: Ellipsoidally Symmetric Solvent Approximation

An example of a 3-dimensional histogram of the density,  $g(r, \cos \theta, \phi)$ , is shown in figure 2.4. Each individual plot is a 2-dimensional plot of density as a function of  $r$  and  $\cos \theta$  at a given  $\phi$ ,  $g(r, \cos \theta; \phi)$ .

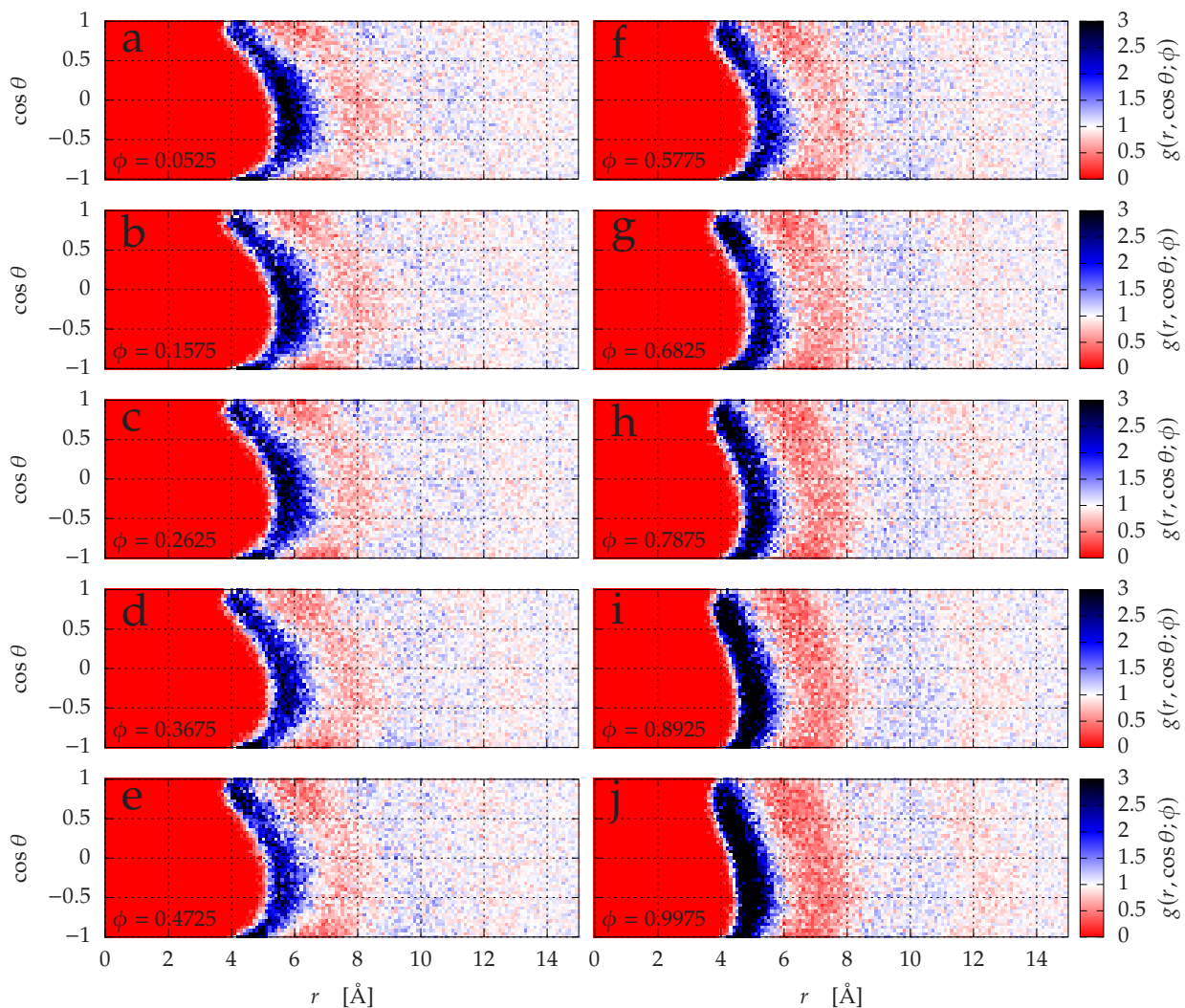
Each individual  $g(r, \cos \theta; \phi)$  plot has the distance  $r$  on the abscissa and the polar tilt  $\cos \theta$  on the ordinate. So the top row of pixels is the  $g(r)$  of chloroform when the hydrogen is pointing towards the solute. Each subsequent row is the  $g(r)$  at a different polar tilt until we reach the bottom row which is the  $g(r)$  of chloroform when the hydrogen is pointing away from the solute.

The general oblate shape of chloroform can be seen from these plots by the bulge of the first solvation shell enhancement as a function of  $\cos \theta$ . For example, a perfectly spherical solvent would have the exact same  $g(r)$  for all  $\cos \theta$  and  $\phi$ , so all the  $\phi$  plots would look identical each with the density enhancement (dark blue stripe) straight up and down.

It can be seen that  $\phi = 0$  radians (figure 2.4a) corresponds to one of the chlorine atoms pointing towards the solute whereas  $\phi = \pi/3$  (figure 2.4j) corresponds to the solute being half way between two chlorines. The first solvation shell bulges more as a function of  $\cos \theta$  when  $\phi = 0$  radians because the chlorine is between the solute and the carbon atom of the solvent. However, as  $\phi$  increases the bulge of the first solvation shell becomes less prominent because the solute is able to wedge itself in-between the two chlorines.

**Implementation Considerations:** First, the histograms are analytically extrapolated, (section 2.2.2.6), to small distances until the histograms reach a predetermined cut-off value which was  $\pm 1000$  in the case of all simulations shown here. After the analytically extrapolated values have been added the histograms require some sort of interpolation. Cubic spline interpolation is used to capture the curvature that is especially important in regions of quick change in the underlying function where linear interpolation would overestimate or underestimate a function's value. To that point, the logarithm of the distribution function is interpolated instead of the distribution function itself because  $\ln g(r; \cos \theta, \phi)$  varies more gradually than  $g(r; \cos \theta, \phi)$  and interpolation of a more gradually varying function is more accurate.

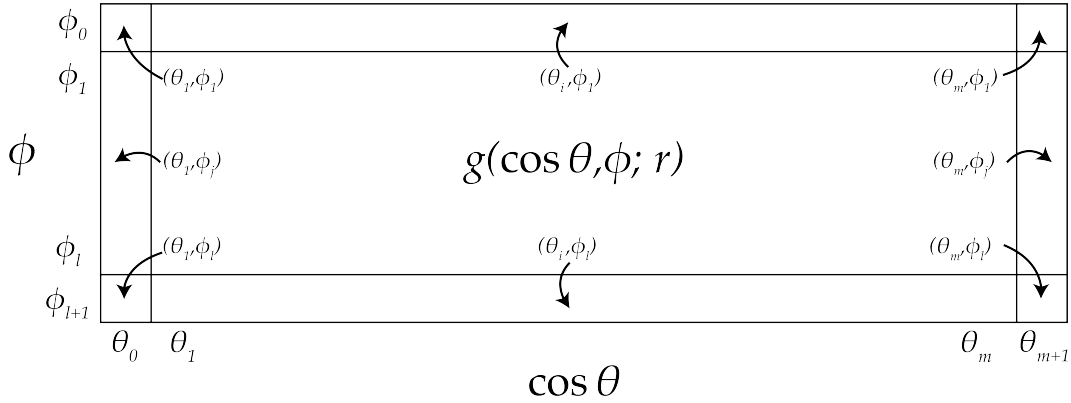
Calculating the second derivatives required for cubic spline interpolation requires knowledge of the first derivatives at both endpoints,  $y'_1$  and  $y'_n$ . Since we are interpolating  $\ln g(r; \cos \theta, \phi)$  the derivative at small distance,  $y'_1$ , will be the value of the force at that distance which is something that is already calculated during the analytic extrapolation. The first derivative at the long distance endpoint,  $y'_n$ , will be zero.



**Figure 2.4:** The distribution function  $g(r, \cos \theta, \phi)$  between chloroform and a spherical solute particle with  $r_{\min} = 5.0 \text{ \AA}$  and charge  $0.0 q_e$  as a function of  $r$ ,  $\cos \theta$ , and  $\phi$ .  $\phi = 0$  radians corresponds to a twist where a chlorine atom is pointing towards the solute, whereas  $\phi = \pi/3$  radians corresponds to the solute being halfway between two chlorines.

In the model system in IS-SPA, section 2.2.3, a solvent is placed in a certain cell and then oriented within that cell such that it samples all orientations relative to each solute. The solute–cell displacement distance,  $r$ , is interpolated first for all  $(\cos \theta, \phi)$  arrays which results in a 2-dimensional array for each solute, e.g.  $g_1(\cos \theta, \phi; r)$  and  $g_2(\cos \theta, \phi; r)$ . The  $\cos \theta$  and  $\phi$  dimensions are splined using `symm_tridag` subroutine in code 2.9 since the data along  $\cos \theta$  and  $\phi$  are symmetric about their endpoints e.g.  $\cos \theta_0 = \cos \theta_1$ ,  $\cos \theta_{m+1} = \cos \theta_m$ , and  $\phi_0 = \phi_1$ ,  $\phi_{l+1} = \phi_l$  where the actual sampled data spans  $1-m$  and  $1-l$  for  $\cos \theta$  and  $\phi$  respectively; this is visually depicted in figure 2.5. This process is performed in the `set_tmp_arrays` subroutine of code 2.7. Once the 2-dimensional

$(\cos \theta, \phi; r)$  plane has been splined along  $\cos \theta$  and  $\phi$  the relevant values are interpolated and bicubic interpolation is used to determine the unknown value.



**Figure 2.5:** Depiction of symmetry of temporary array created for a cell in a 3-dimensional IS-SPA calculation. Since both  $\cos \theta$  and  $\phi$  histograms span a half-period range, the array is symmetric about the boundary.

### 2.2.2.3 2-Dimensional Histograms: Spheroidally Symmetric Solvent Approximation

If we assume that the variability in the density and force is insignificant in the  $\phi$ -dimension then we can instead represent chloroform with 2-dimensional histograms i.e. approximate chloroform as a spheroid for the density and force by averaging the functions over  $\phi$  in the following manner,

$$g(r, \cos \theta) = \frac{\int d\phi g(r, \cos \theta, \phi)}{\int d\phi} \quad (2.4)$$

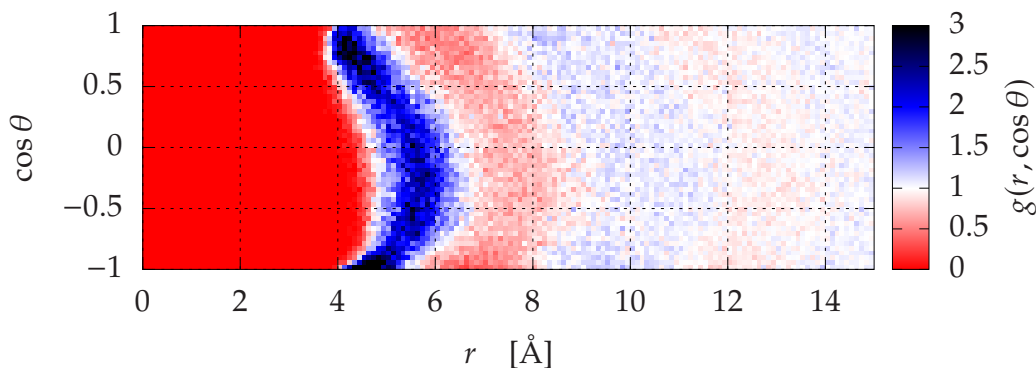
for the distribution function and,

$$f(r, \cos \theta) = \frac{\int d\phi g(r, \cos \theta, \phi) f(r, \cos \theta, \phi)}{\int d\phi g(r, \cos \theta, \phi)} \quad (2.5)$$

for the force, where  $g(r, \cos \theta, \phi)$  is the weighting term in the weighted average of the force in equation (2.5).

In other words, in the 2-dimensional case chloroform is approximated as an oblate spheroid. An oblate spheroid has two unique axes, the polar axis which is shorter and the degenerate equatorial axes which are longer.

The resultant distribution function in figure 2.6 has the intensity of its first solvation shell enhancement dispersed relative to the individual  $\phi$  plots in figure 2.4 for medial values of  $\cos \theta$  where the  $\phi$ -distributions in figure 2.4 differ and concentrated for extreme values of  $\cos \theta$  where they are similarly invariable in  $\phi$ .



**Figure 2.6:**  $g(r, \cos \theta)$  shows the ellipticity of chloroform as a function of  $\theta$  for an  $r_{\min} = 5 \text{ \AA}$  and  $0.0 q_e$  charge solute.  $\cos \theta = 1$  is when the hydrogen is pointing towards the solute, and  $\cos \theta = -1$  is when the hydrogen is pointing away from the solute.

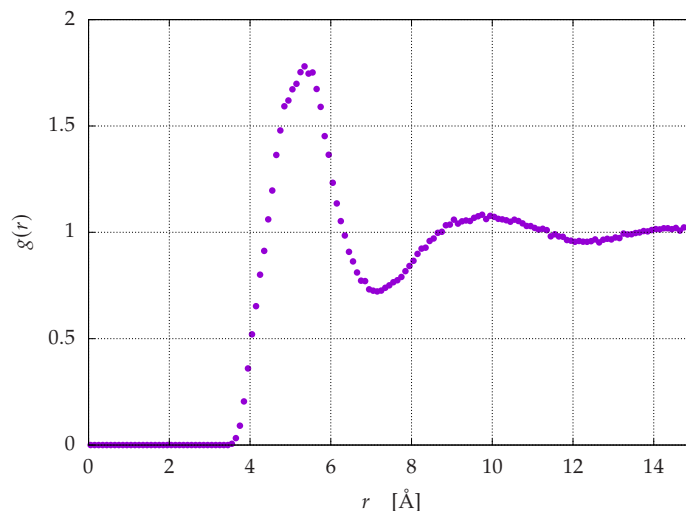
**Implementation Considerations:** For each cell in the model system (section 2.2.3) the polar-tilt orientation,  $\cos \theta$ , must be sampled for both solutes. The solute–cell displacement,  $r$ , is spline-interpolated first for each  $\cos \theta$  array which results in a 1-dimensional array for each solute, e.g.  $g_1(\cos \theta; r)$  and  $g_2(\cos \theta; r)$ . This  $\cos \theta$  array is then interpolated via cubic spline interpolation using `symm_tridag` in code 2.9 because the data along  $\cos \theta$  is symmetric about the endpoints such that  $\cos \theta_0 = \cos \theta_1$  and  $\cos \theta_{m+1} = \cos \theta_m$  where the actual histograms in  $\cos \theta$  span from 1 to  $m$ .

#### 2.2.2.4 1-Dimensional Histograms: Spherically Symmetric Solvent Approximation

To simplify even further we can average the density and force over  $\cos \theta$  and  $\phi$ ,

$$g(r) = \frac{\int d\cos \theta \int d\phi g(r, \cos \theta, \phi)}{\int d\cos \theta \int d\phi} \quad (2.6)$$

$$f(r) = \frac{\int d\cos \theta \int d\phi g(r, \cos \theta, \phi) f(r, \cos \theta, \phi)}{\int d\cos \theta \int d\phi g(r, \cos \theta, \phi)} \quad (2.7)$$



**Figure 2.7:** Chloroform distribution function  $g(r)$  around  $r_{\min} = 5 \text{ \AA}$  and charge  $0.0 q_e$  solute averaged over  $\cos \theta$  and  $\phi$ .

Representing chloroform with equations (2.6) and (2.7) is analogous to approximating chloroform as a sphere i.e. the only important variable in modeling the solvent density and force is the distance between it and the solute atom,  $r$ . This is referred to as approximating the solvent as a sphere because a sphere has infinite rotational symmetry, i.e. there is no change in the sphere upon rotation about its center, only position matters. If sufficiently accurate, modeling chloroform as a sphere would be ideal because the sampling and interpolation of a 1-dimensional histogram is much simpler and more efficient than using higher dimensional histograms. The resultant distribution function is shown in figure 2.7.

**Implementation Considerations:** Since all of the orientational degrees of freedom are averaged over when treating chloroform as a sphere, the only interpolation used in the case of 1-dimensional histograms is spline interpolating the solute–cell displacement,  $r$ , with the `splint` subroutine in code 2.9.

### 2.2.2.5 Cubic Spline Interpolation

Cubic splines were chosen for interpolating the histogram data because they can be efficiently calculated and interpolated on the fly. Numerical Recipes<sup>13</sup> also has a cubic spline subroutine as

<sup>13</sup>Press, W. H. et al., *Numerical Recipes in Fortran 77: The Art of Scientific Computing*, Second; Cambridge England: 1996; Vol. 1.



well as a spline interpolation subroutine already written in fortran which was altered for use in this work. A spline function is composed of a polynomial between each pair of data points with continuous derivatives across the entire spline where the level of derivative that retains continuity is determined by the order of the spline. Cubic splines are popular because they produce an interpolated function that is continuous in the second derivative and smoothly varying in the first derivative.

The specifics of how spline interpolation works is covered in great detail in *Numerical Recipes in Fortran 77*.<sup>13</sup> However, in short, with knowledge of the points in the dataset as well as the slopes of the function at the two extremes, a set of equations is obtained for the values and the second derivatives at each data point. Solving this set of equations results in a set of second derivatives for each data point given. The spline functions are linear in the second derivative and cubic in the 0-order derivative.

For data sets that exhibit some symmetry the spline equations can be altered slightly. Instead of providing the first derivative at the extremes of the input data set, if the function is symmetric such that the function and its derivatives at the 1<sup>st</sup> point would be identical to those at the 0<sup>th</sup> point and the value and derivative at the  $N^{\text{th}}$  and  $(N + 1)^{\text{th}}$  would also be identical, the equations can be altered such that no first derivatives are required. This fact was exploited when running IS-SPA of chloroform when the solvent was modeled with 2 and 3 degrees of freedom,  $r$ ,  $\cos \theta$ , and  $\phi$ , and can be seen in the fortran subroutines `symm_spline` and `symm_splint` as compared to the general non-symmetric subroutines `spline` and `splint` all contained in code 2.9.

Another direct result of histograms being measured from simulation data is that they do not have value in the regions where there was no sampling. Given infinite sampling time and infinitesimal bin sizes the histograms would smoothly transition from regions of plentiful sampling to those of negligible sampling. However, since our simulation data and thus our histograms do not have this quality we have to find a way of continuing the histogrammed data into those regions in a manner that is physically motivated. These smooth transitions are important because the forces that are imparted in regions of poor sampling, like small separations distances  $r$ , are large and thus have

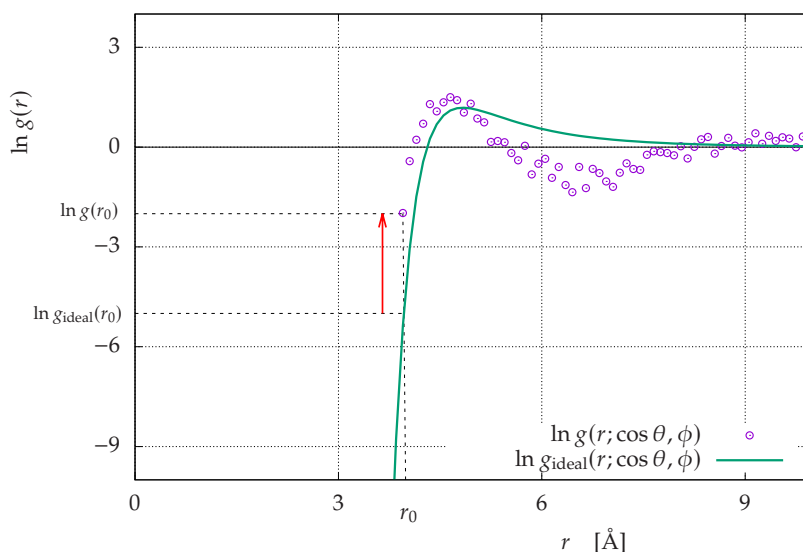
---

<sup>13</sup>Press, W. H. et al., *Numerical Recipes in Fortran 77: The Art of Scientific Computing*, Second; Cambridge England: 1996; Vol. 1.

a large effect on the physics when they occur. We call this process of smoothly transitioning the histogrammed data, analytic extrapolation.

### 2.2.2.6 Analytic Extrapolation of Histograms

In all of the IS-SPA codes that use histogram input for  $g(r)$  and  $f(r)$  there are issues of poor sampling near the solute particles. At small  $r$  the distribution function goes to 0, (or equivalently  $\ln g(r)$  goes to  $-\infty$ ), and the LJ force goes to  $+\infty$ . In an effort to smoothly transition these functions to their respective end-behaviors in a rigorous and physically motivated manner we implemented a process that we refer to as analytic extrapolation. Analytic extrapolation is the process by which we stitch together the measured histogram data with idealized data from an ab initio calculation of a single solute and solvent in vacuum.



**Figure 2.8:** Illustration of analytic extrapolation for a distribution function in  $r$  at a particular  $\cos \theta$  and  $\phi$ . For all  $r$  bins smaller than  $r_0$  the shifted ideal distribution is used where the shift is shown by the red arrow.

The idealized data is the direct distribution and force of a single solute and solvent as a function of  $r$ ,  $\cos \theta$ , and  $\phi$  matching the same  $r$ ,  $\cos \theta$ , and  $\phi$  of the histogrammed data. At small  $r$  when the measured data has no sampling, the ideal data is added to create a smooth transition to the end-behavior. This process is done to the 3-dimensional histograms before any averaging over  $\phi$  or  $\cos \theta$  occurs. Once the functions are analytically extrapolated the weighted averages in

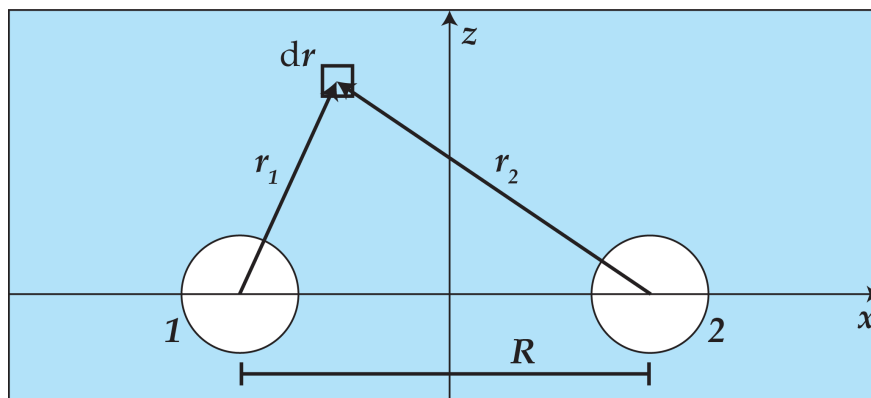
sections 2.2.2.3 and 2.2.2.4 over  $\cos \theta$  and  $\phi$  are performed. This is done in the `spline_hist_array` subroutine of the various IS-SPA fortran codes 2.5 and 2.8.

Figure 2.8 shows an example of a measured distribution function histogram  $\ln g(r; \cos \theta, \phi)$  with its corresponding ideal distribution function  $\ln g_{\text{ideal}}(r; \cos \theta, \phi)$ . All values to the left of the first bin with a count in it,  $r_0$ , are given the value of the ideal distribution shifted by  $\ln g(r_0) - \ln g_{\text{ideal}}(r_0)$ , (red arrow), such that the shifted ideal distribution function passes through  $\ln g(r_0)$ . These techniques are used in all of the IS-SPA fortran codes to varying extents.

To test our approximations outlined above we have used two systems: the model system and the molecular system. Each system has been simulated using AESMD and IS-SPA. In the model system the IS-SPA results are obtained by numerically integrating equation (2.2). For the molecular system the IS-SPA results are generated via Monte Carlo integration.

### 2.2.3 Model System

To test our approximations we use the model system depicted in figure 2.9. This system was simulated explicitly with AMBER 16<sup>7</sup> via umbrella sampling (US) simulations as well as replicated with various levels of IS-SPA theory.



**Figure 2.9:** Schematic of the simple model system that was used to compare IS-SPA and AESMD.

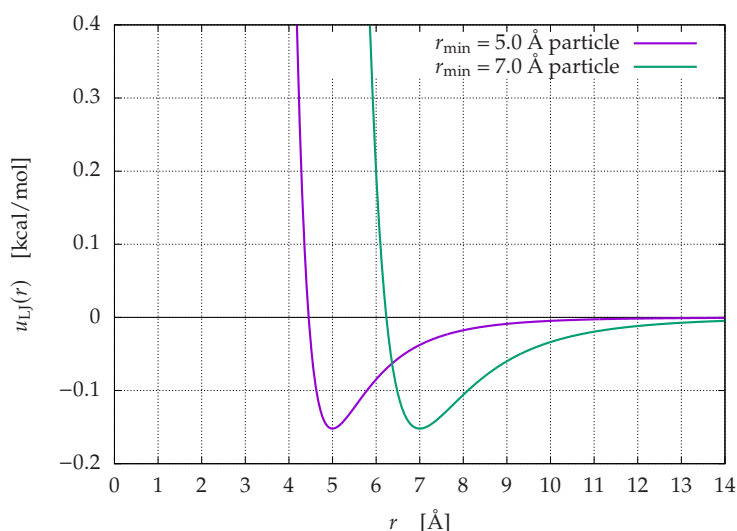
The model system is a pair of custom made solutes in the chloroform-box solvent included in AMBER 16.<sup>14</sup> We created the spherical solute particles with LJ potential parameters of  $\epsilon = 0.152$  kcal/mol and  $r_{\text{min}} = 5 \text{ \AA}$  and  $7 \text{ \AA}$  as shown in equation (2.8) and figure 2.10.

<sup>7</sup>Case, D. et al. AMBER16 Package.

<sup>14</sup>Cieplak, P.; Caldwell, J.; Kollman, P. *Journal of Computational Chemistry* **2001**, *22*, 1048–1057.

$$u_{\text{LJ}}(r) = \epsilon \left[ \left( \frac{r_{\text{min}}}{r} \right)^{12} - 2 \left( \frac{r_{\text{min}}}{r} \right)^6 \right] \quad (2.8)$$

where the LJ potential,  $u_{\text{LJ}}$ , is shown in terms of the potential well minimum distance,  $r_{\text{min}}$ , and the depth of the well,  $\epsilon$ , as a function of separation distance between the particles,  $r$ . The value of  $\epsilon$  used is the same as that of an oxygen atom in the generalized AMBER force field (GAFF).<sup>15</sup> The values for  $r_{\text{min}}$  were chosen to reflect the size of a small general solute rather than being representative of a particular chemical species. All subsequent references to *explicit* results in this section are referencing the AESMD simulations of this model system. In later experiments the solutes were given equal and opposite charges ranging from  $0.0 q_e$  to  $1.0 q_e$  in  $0.1 q_e$  increments, where  $q_e$  is the elementary charge, with solute **1** positively charged and solute **2** negatively charged.



**Figure 2.10:** Potential energy for the two custom-made spherical solutes with no charge using the LJ potential energy in equation (2.8).

This model system was chosen because it is the simplest system for which we can calculate a PMF to compare IS-SPA results to the explicit ones IS-SPA is trying to replicate. The solute particles were given radii larger than a typical atom because the majority of interesting solutes that will be studied will be on this size scale or larger and smaller solute particle size may lead to problems arising from the relative scale of the solute and solvent being nearly the same. The solutes are spherical to simplify the analysis of the system by taking any orientational degrees of

<sup>15</sup>Wang, J. et al. *Journal of Computational Chemistry* **2004**, *25*, 1157–1174.

freedom away from the solute, thus allowing us to focus only on the orientation of the solvent when pertinent.

## 2.2.4 Measuring Molecular Dynamics Simulation

IS-SPA is a bottom-up approach implicit-solvation model. This means that it tries to replicate the microscopic properties of the solvent. To build the distribution function and force function histograms for equation (2.2) for the solvent, the explicit-solvent from AESMD simulations must be studied first.

Measurements from AESMD simulations are required for the model system to test the solvent distribution and force function histograms. Once the model system mean force calculations are complete the chemical system must be fit to and chemical system simulations can be run using the fitted atomic distribution and force histograms.

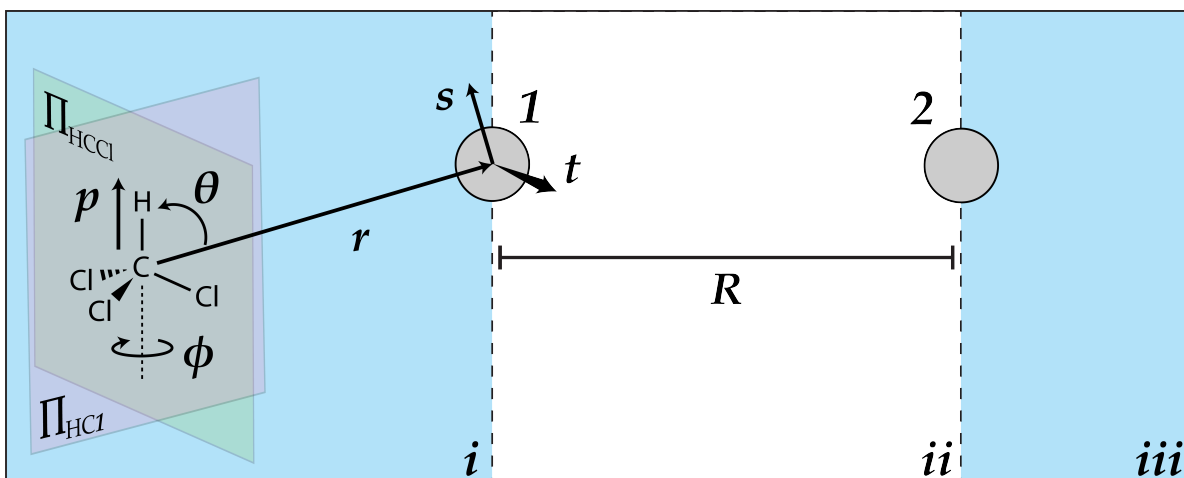
### 2.2.4.1 Model System Measurements

The model system pictured in figure 2.9 must be run with the explicit all-atom version of the solvent that you wish to replicate, which in the case of this work is the built-in AMBER chloroform box.<sup>14</sup> Measurements of this system were taken with the homebuilt code in appendix 2.1 which collects 3-dimensional histograms of the ellipsoidally symmetric distribution and force functions of chloroform for both solutes in the model system independently,  $g_i(|\mathbf{r} - \mathbf{R}_i|, \cos \theta, \phi)$  and  $f_{i,\text{solv}}(|\mathbf{r} - \mathbf{R}_i|, \cos \theta, \phi)$ . Only solvent molecules on the far side of the solutes are measured to reduce the effect of the other solute on the solvent e.g. for solute **1** only solvents in the shaded region on the left of figure 2.11 are measured and vis versa for solute **2**.

The code 2.1 works by looping through all the solvent molecules in all frames of the MD simulation and binning them into  $(r, \cos \theta, \phi)$  bins for the relevant solute. First, the solvent is determined to be in one of the three zones (**i**, **ii**, **iii**) in figure 2.11. Then, depending on the zone, the solvent distance to the relevant solute is measured and binned into the histogram for (**i**, **iii**) or the solvent is skipped for **ii**. The  $\cos \theta$ -bin is determined by calculating the dot-product of  $\hat{\mathbf{p}}$  and  $\hat{\mathbf{r}}$ . The  $\phi$ -bin is determined by calculating the angle between the normal vectors of the planes  $\Pi_{\text{HCCI}}$

---

<sup>14</sup>Cieplak, P.; Caldwell, J.; Kollman, P. *Journal of Computational Chemistry* **2001**, *22*, 1048–1057.



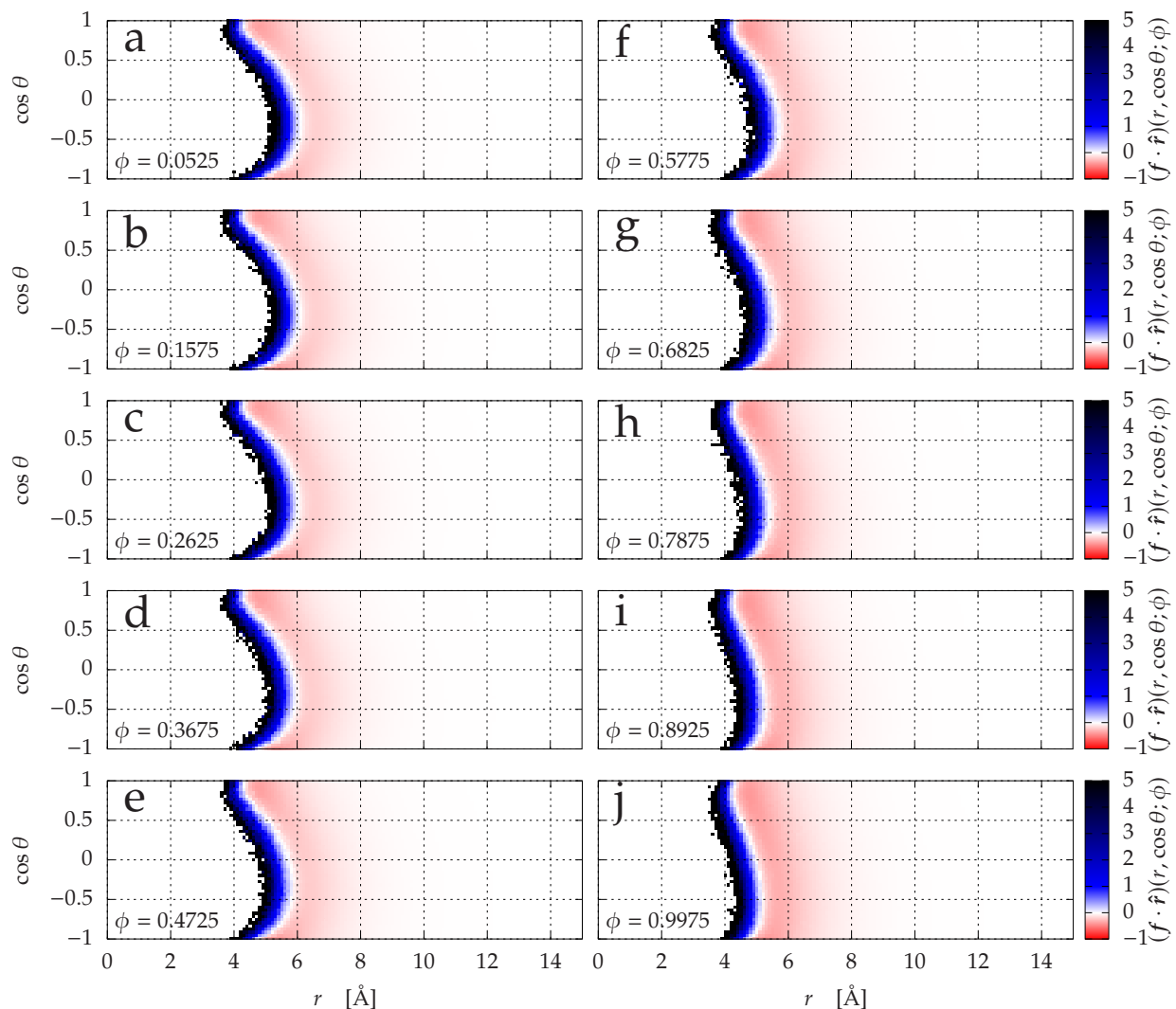
**Figure 2.11:** Schematic showing the measurement of the model system explicit MD simulation. A solvent is shown in the sampling region, *i*, for solute **1** with all of the solvent vectors, angles, and planes necessary for the calculation of  $r$ ,  $\cos \theta$ , and  $\phi$ . The sampling region *iii* is used for solute **2** and the region *ii* is ignored.

and  $\Pi_{\text{HCl}}$  and wrapping it into the  $[0: \frac{\pi}{3}]$  range. Measuring the angle between the planes will give the correct  $\phi$  regardless of the  $\cos \theta$  tilt.

The 3-dimensional force histogram from each solvent is measured by calculating the sum of the forces from all five atoms of the solvent and projecting them along the three orthogonal solvent vectors,  $r$ ,  $s$ , and  $t$ . The result is three force histograms:  $f_r(|r - R_i|, \cos \theta, \phi)$ ,  $f_s(|r - R_i|, \cos \theta, \phi)$ , and  $f_t(|r - R_i|, \cos \theta, \phi)$  where  $f_r$  represents the force projection along  $\hat{r}$  i.e.  $(f \cdot \hat{r})$ ,  $f_s$  the projection along  $\hat{s}$  i.e.  $(f \cdot \hat{s})$ , and  $f_t$  the projection along  $\hat{t}$  i.e.  $(f \cdot \hat{t})$ . In subsequent codes these forces are averaged into lower dimensional force histograms. Finally, the force histograms are averaged by dividing by the number of counts in each  $(r, \cos \theta, \phi)$  bin and the distribution histogram is volume corrected and normalized.

If the solvent is being modeled with the 2 or 3-dimensional distribution and force functions, there can be appreciable components of the force that do not lie solely along  $r$ , especially at close separation distances. To illustrate this fact, figures 2.12, 2.13, and 2.14 are the components of the force that lie along  $r$ ,  $s$ , and  $t$  respectively.

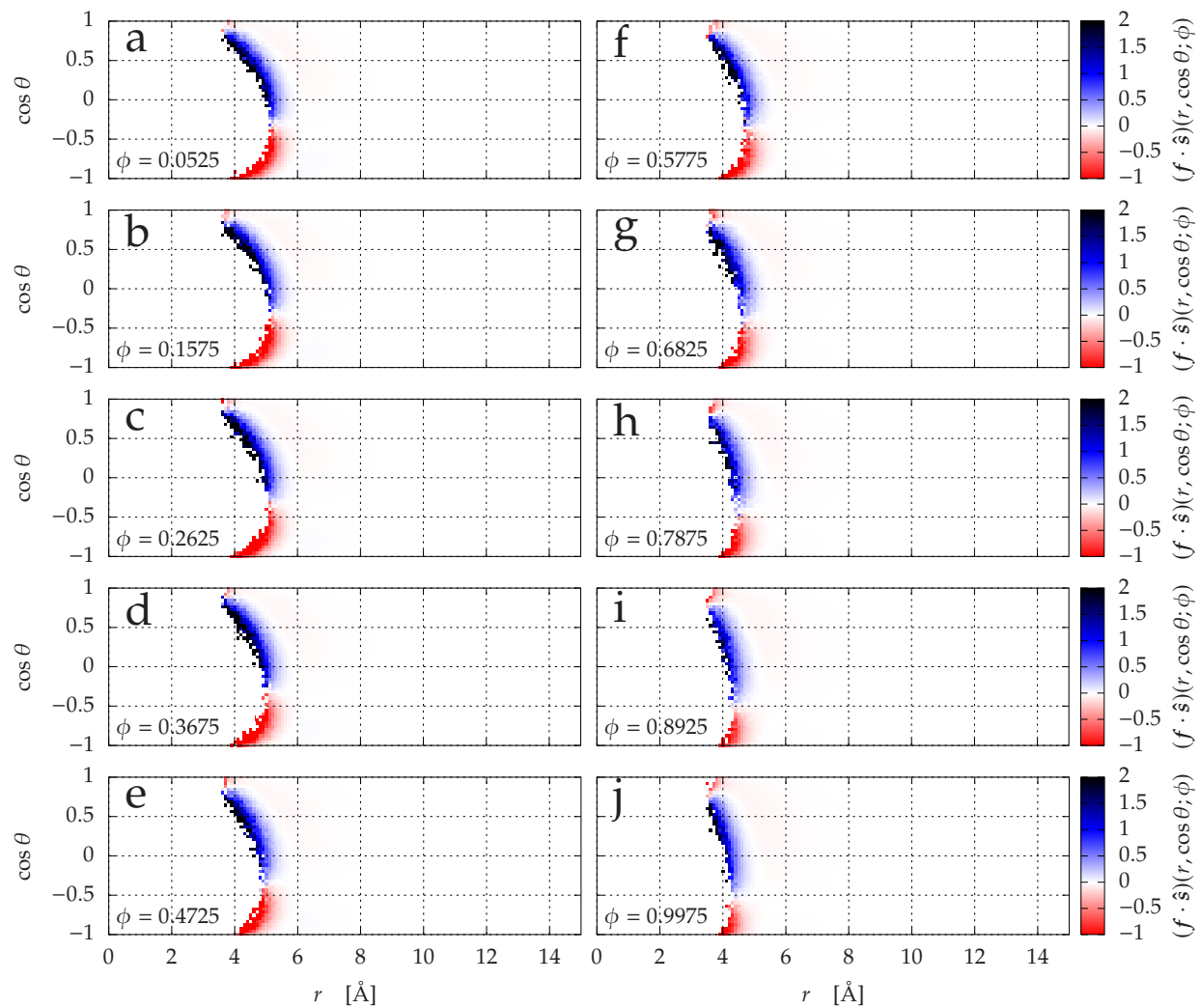
The force along  $r$  behaves similarly for all  $\cos \theta$  in that it transitions from attractive at large  $r$  to repulsive at short  $r$ . This general behavior is characteristic of a LJ force between two atoms. The ellipsoidal symmetry of chloroform can be seen in figure 2.12 just as was described in section 2.2.2.2 for figure 2.4.



**Figure 2.12:** The 3-dimensional force projected along  $r$ , the separation vector of solvent carbon atom and solute center. The blue/black forces are repulsive between solvent and solute, the red bins are attractive.

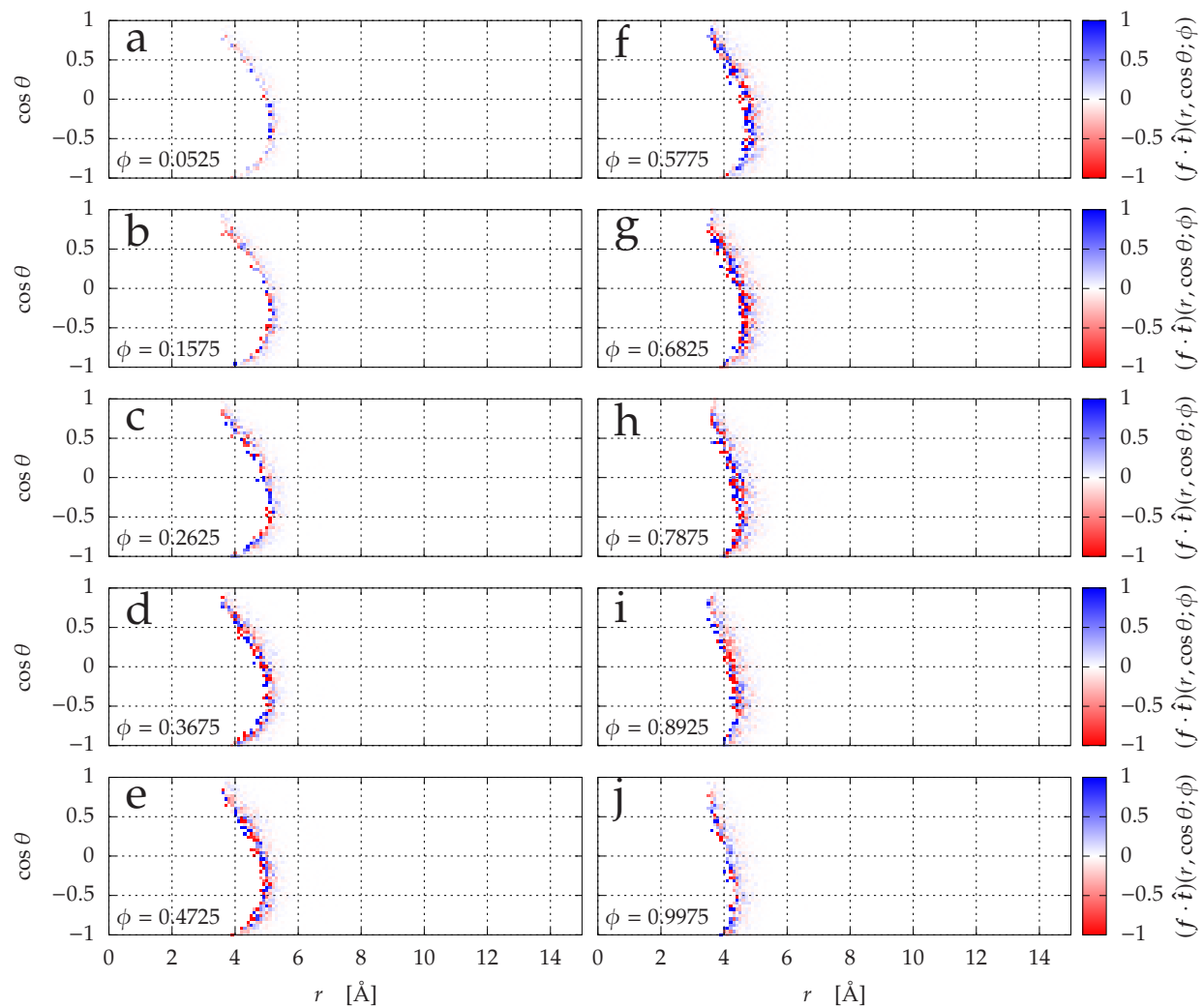
The force along  $s$  behaves differently as a function of  $\cos \theta$ . The direction of  $s$  is determined in code 2.1 by  $t \times r$  which results in  $s$  always being orthogonal to  $r$  and  $t$  and in the plane of  $p$  and on the same side of  $r$  as  $p$ . So when  $\cos \theta$  crosses zero the direction of  $s$  flips. This is what gives  $f_s$  the sign flipping quality as a function of  $\cos \theta$ . The fact that the hydrogen is smaller than the chlorines vertically offsets that sign change along  $\cos \theta$ .

The force along  $t$  comes from one of the chlorines being closer to the solute than another and impinging a force orthogonal to  $r$  as well as the polar plane that  $s$  resides in.



**Figure 2.13:** The 3-dimensional force projected along  $s$ , the solvent vector orthogonal to  $r$  and in the plane of  $p$ . The blue/black forces correspond to the solute being pushed along  $s$ , the red bins are forces along  $-s$ .



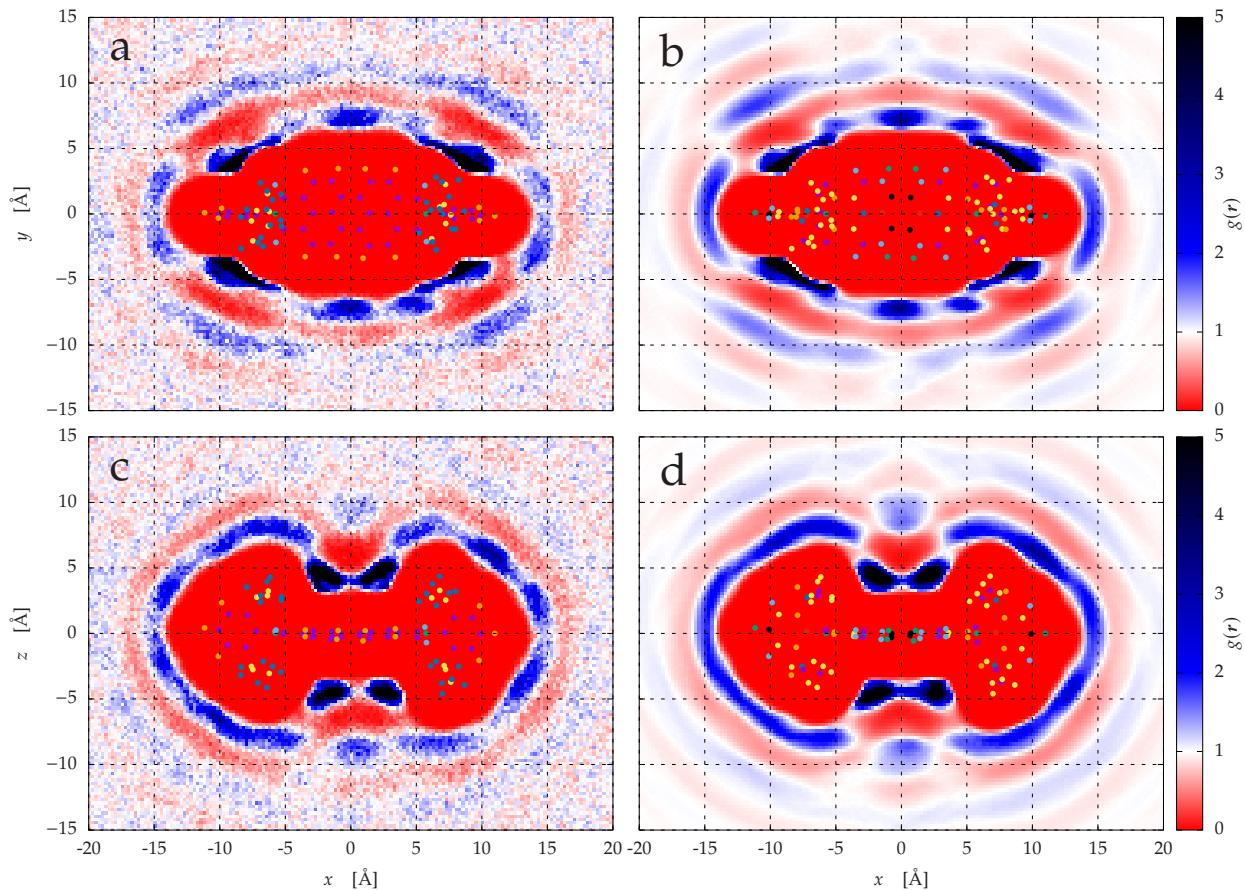


**Figure 2.14:** The 3-dimensional force projected along  $\hat{t}$ , the third solvent vector orthogonal to both  $r$  and  $s$ .

#### 2.2.4.2 Chemical System Measurements

Once the level of theory for the solvent has been tested on the model system and has reproduced the mean force or PMF sufficiently it is time to move on to a chemical system. For my work, the solute chosen was a perylene diimide derivative called Lumogen Orange (LO). LO AESMD simulations were run with the same explicit chloroform solvent model as the model system. The PMF of dimerization was measured via umbrella sampling AESMD simulations. AESMD belly simulations, where the atoms of the solute and solvent are effectively frozen in place by setting their masses to be  $1 \times 10^9$  g/mol, were also performed for a LO monomer. The umbrella sampling MD simulations of the dimer allow for the measuring of the PMF and mean force of dimerization. The belly simulation of the monomer allows for the fitting of the atomic distribution and force histograms of the solvent about the fixed solute atoms.

To calculate the atomic distribution functions to be used in equation (2.3) we first numerically map the molecular distribution function using the belly simulation to histogram a 3-dimensional grid of the simulation box about the fixed LO molecule. After the explicit molecular distribution function has been collected, via belly simulation and 3D binning script 2.2, it is fitted with  $N$  individual radial distribution function histograms centered on each of the solute atoms. Two example slices of the 3-dimensional density grid for LO are shown in figure 2.15a,c. This is the data that is fit to using codes 2.3 and 2.4. The resulting fits are shown in figure 2.15b,d.



**Figure 2.15:** (a,b)  $xy$ -plane, and (c,d)  $xz$ -plane slices of the 3-dimensional solvent distribution function  $g(\mathbf{r}; \mathbf{R}^N)$  of LO from (a,c) AESMD, and (b,d) the histogram fit using equation 2.3. The bins are  $0.25 \text{ \AA}$  side-length cubes and the solute atoms are colored by (a,c) their GAFF atom types in MD simulation, and (b,d) their symmetric atom type for histogram fitting.

## 2.2.5 Fitting Histograms

The histogrammed explicit box data shown in figure 2.15a,c is used to fit the product of radial distribution functions for each solute atom resulting in the fitted molecular distribution function in figure 2.15b,d. The fitting is performed by codes 2.3 and 2.4. The following is a brief description of how the fitting is accomplished.

If you have an analytic function with three parameters then you need to fit  $3N$  parameters for the molecule in IS-SPA, where  $N$  is the number of unique atom types in the molecule. With a histogram this gets much worse. You need to fit a  $\lambda_i$  for every bin in the histogram which gets to be an intractable number of parameters quickly especially when considering multidimensional histograms.

$$\lambda(r) = \sum_i^N \lambda_i [\Theta(r - r_i) - \Theta(r - r_i + dr)] \quad (2.9)$$

where,  $\Theta(r)$  is the Heaviside step function. Equation (2.9) is essentially saying that the histogram fit function,  $\lambda(r)$ , is composed of  $N$  independent points,  $\lambda_i$ , that each span their respective domains from  $r - r_i$  to  $r - r_i + dr$ .

In practice this was done by choosing a number of unique atom types in the solute and fitting to the results from code 2.2 which are plotted in figure 2.15a,c. So for a given atom type each  $\lambda_i$  is being adjusted to best fit its respective spherical shell around all atoms of that type. The fitting metric used for the first guess was  $\chi^2$ ,

$$\chi^2 = \sum_i \frac{(y_i - f(x_i; \alpha))^2}{\sigma_i^2} \quad (2.10)$$

where the sum is over all data points  $y_i$ , which are the cells in our 3-dimensional distribution file in figure 2.15a,c.  $f(x_i; \alpha)$  is the fit function at the location of the  $i^{\text{th}}$  cell,  $x_i$ , given the values of all the histograms of all the solute atom types at that location. Each term of the sum is weighted by the variance of the data point  $\sigma_i$  assuming Poissonian statistics. The Poisson distribution, equation 2.11, is used for determining the variance because the data we are fitting to is derived from counting measurements.

$$P(N; \lambda) = \frac{\lambda^N e^{-\lambda}}{N!} \quad (2.11)$$

where  $P(N; \lambda)$  is the probability of observing the value  $N$  given the average observation value  $\lambda$ . For the Poisson distribution in the classical inference the average observed value of  $N$ ,  $\langle N \rangle$ , is unsurprisingly the distribution average  $\lambda$ , which can be proven by solving for  $\langle N \rangle$  in,

$$\begin{aligned} \langle N \rangle &= \sum_{N=0}^{\infty} N P(N; \lambda) \\ &= \lambda \end{aligned} \quad (2.12)$$

and the variance of  $N$  is equal to  $\lambda$  as well which can be found by solving,

$$\begin{aligned}
\langle N^2 \rangle - \langle N \rangle &= \left( \sum_{N=0}^{\infty} N^2 P(N; \lambda) \right) - \left( \sum_{N=0}^{\infty} N P(N; \lambda) \right)^2 \\
&= (\lambda^2 - \lambda) - (\lambda)^2 \\
&= \lambda
\end{aligned} \tag{2.13}$$

However, for the data we are looking at, we are given a measurement  $N$  and we want to know what the average value of the distribution function  $\lambda$  is given that measurement, this is called Bayesian inference

### 2.2.5.1 Bayesian Inference

The probability of the distribution having the average  $\lambda$  given an observation from that distribution,  $N$ . In other words, you have this underlying probability distribution and you take a measurement of it and get a value,  $N$ . Given that value  $N$ , what is the probability that the average of the distribution is  $\lambda$ ? The probability from equation (2.11) is the same but the variables flip.

$$P(\lambda; N) = \frac{\lambda^N e^{-\lambda}}{N!} \tag{2.14}$$

With this distribution we want to derive the mean distribution-average  $\langle \lambda \rangle$ , and the variance of that distribution-average  $\text{Var}(\lambda)$ . It is important to note that while  $N$ , the number of counts generally or in our case the measured  $g(r)$  which is related to the number of counts, is a discrete variable, the average-value  $\lambda$  is a continuous variable. Therefore, when  $N$  is the variable sums will be used, as in equations (2.12) and (2.13), and when  $\lambda$  is the variable integrals will be used to reflect the discrete or continuous nature of the variables respectively. The mean can be derived by the continuous analogue of equation (2.12),

$$\begin{aligned}
\langle \lambda \rangle &= \int_0^{\infty} d\lambda \lambda P(\lambda; N) \\
&= (N + 1)
\end{aligned} \tag{2.15}$$

where the  $N$  is the measurement we are given from the data set e.g. the number of counts in a given bin or in our case the measured  $g(r)$  from MD simulation. Somewhat surprisingly we get that, given a measurement of  $N$ , the mean average-value is  $(N + 1)$ . For the variance of  $\lambda$  we solve the continuous analogue of equation (2.13),

$$\begin{aligned}\langle \lambda^2 \rangle - \langle \lambda \rangle^2 &= \left( \int_0^\infty d\lambda \lambda^2 P(\lambda; N) \right) - \left( \int_0^\infty d\lambda \lambda P(\lambda; N) \right)^2 \\ &= (N + 2)(N + 1) - (N + 1)^2 \\ &= (N + 1)\end{aligned}\tag{2.16}$$

However, for the histogram fitting codes 2.3 and 2.4 we are fitting the distribution function  $g(r)$  which is related to the free energy by,

$$\begin{aligned}A &= -T \ln g(r) \\ &= -T \ln \lambda\end{aligned}\tag{2.17}$$

where  $T$  is the temperature in units of energy. The substitution in the second line of equation (2.17) can be made because the single measurement of the distribution function,  $N$ , is sampled from the underlying distribution with average,  $\lambda$ , and the free energy is determined by the average not a single measurement. We can substitute this into the posterior probability (2.14), and then solve for the mean free energy  $\langle A \rangle$ ,

$$\langle A \rangle = \int_{-\infty}^{\infty} dA A P(A; N)\tag{2.18}$$

where the limits of integration reflect the fact that the Helmholtz free energy,  $A$ , can span the range  $(-\infty, \infty)$ . We can simplify the integral in terms of its limits by changing the variable of integration to be  $g(r)$  or  $\lambda$  which only spans  $[0, \infty)$ .

$$\begin{aligned}
\langle A \rangle &= \int_0^{\infty} d\lambda A(\lambda) P(\lambda; N) \\
&= -\frac{T}{N!} \int_0^{\infty} d\lambda \ln \lambda \lambda^N e^{-\lambda}
\end{aligned} \tag{2.19}$$

Then using integration by parts we arrive at,

$$\langle A \rangle_N = T \left( \gamma - \sum_{k=1}^N k^{-1} \right) \tag{2.20}$$

where  $\gamma$  is the Euler-Mascheroni constant defined by,

$$\begin{aligned}
\gamma &\equiv \lim_{x \rightarrow \infty} \left[ -\ln x + \sum_{k=1}^x k^{-1} \right] \\
&= - \int_0^{\infty} dx e^{-x} \ln x
\end{aligned} \tag{2.21}$$

The variance of  $A$  is calculated in the same manner as equation (2.16) with,

$$\begin{aligned}
\langle A^2 \rangle - \langle A \rangle^2 &= \left( \int_0^{\infty} d\lambda A(\lambda)^2 P(\lambda; N) \right) - \left( \int_0^{\infty} d\lambda A(\lambda) P(\lambda; N) \right)^2 \\
&= \left( T^2 \left\{ \frac{\pi^2}{6} - \sum_{k=0}^{N-1} (k+1)^{-2} + \left( \gamma - \sum_{k=0}^{N-1} (k+1)^{-1} \right)^2 \right\} \right) - (\langle A \rangle_N)^2 \\
&= \left( T^2 \left\{ \frac{\pi^2}{6} - \sum_{k=1}^N k^{-2} \right\} + \langle A \rangle_N^2 \right) - (\langle A \rangle_N)^2 \\
&= T^2 \left\{ \frac{\pi^2}{6} - \sum_{k=1}^N k^{-2} \right\}
\end{aligned} \tag{2.22}$$

where  $\langle A \rangle_N$  is the mean free energy given the observation value  $N$  as shown in equation (2.20).

So, in codes 2.3 the result from equation (2.22) is used to calculate the weightings for each term in

$\chi^2, \sigma_i^2$ .

Now that we have the data that we are fitting to we want to minimize  $\chi^2$  in equation (2.10). This is done by finding the root of the first derivative,

$$\frac{\partial \chi^2}{\partial \alpha_{nm}} = \sum_i^{\text{cells}} \frac{2(y_i - f(x_i; \alpha))}{\sigma_i^2} \left( -\frac{\partial f(x_i; \alpha)}{\partial \alpha_{nm}} \right) = 0 \quad (2.23)$$

where the first derivative of the fitting function with respect to one of the fitting parameters  $\alpha_{nm}$  for the  $i^{\text{th}}$  cell is,

$$\begin{aligned} \frac{\partial f(x_i; \alpha)}{\partial \alpha_{nm}} &= \frac{d}{d\alpha_{nm}} \left[ \sum_j^{\text{atoms}} \alpha_{a_j} b_{ij} \right] \\ &= \sum_j^{\text{atoms}} \delta_{na_j} \delta_{mb_{ij}} \end{aligned} \quad (2.24)$$

where  $n$  is the atom type,  $m$  is the distance to the  $i^{\text{th}}$  cell from atom  $n$ ,  $a_j$  is the atom type of atom  $j$ , and  $b_{ij}$  is the distance from atom  $j$  to cell  $i$ . In other words, the elements of the sum in equation (2.24) that are non-zero are the ones where  $n$  is equal to  $a_j$  and  $m$  is equal to  $b_{ij}$ .

The  $(\partial f / \partial \alpha_{nm})$  term, equation (2.24), gives two delta functions which means that whole  $(\partial \chi^2 / \partial \alpha_{nm})$  term only survives when both delta functions are non-zero. This means that  $(\partial \chi^2 / \partial \alpha_{nm})$  only has non-zero value in bins forming cells that fall into spherical shells around atoms of the same type  $n$ .

The fitting result from code 2.3 serves as a first guess for the Poisson regression fitting code 2.4.

### 2.2.5.2 Poisson Regression

Just as in the case of  $\chi^2$  we are minimizing the negative logarithm of a multivariate Gaussian distribution, in Poisson regression we minimize the negative logarithm of a multivariate Poisson distribution,

$$-\ln P(N; \lambda) = \sum_i^{\text{cells}} \ln N_i! + \lambda_i - N_i \ln \lambda_i \quad (2.25)$$

where the  $N_i$ 's are the observed number of counts of a solvent being in a particular cell in the simulation, and the  $\lambda_i$ 's are the parameters of the distribution, the true underlying average number



of counts. We are fitting atomic distribution functions in the form of histograms for each solute atom to these parameters using SPA so we have,

$$\ln \lambda_i = \sum_j^{\text{atoms}} \alpha_{a_j b_{ij}} \quad (2.26)$$

where  $a_j$  is the atom type of atom  $j$ ,  $b_{ij}$  is the distance of the cell  $i$  from that atom  $j$ , and  $\alpha_{a_j b_{ij}}$  is the value of the histogram from atom  $j$  in cell  $i$ .  $\lambda_i$  is akin to the distribution function  $g(r)$  and each  $\alpha$  is the reduced free energy of being at that distance from that atom. In order to minimize equation (2.25) we must take the derivative and set it equal to zero. The derivative is given by,

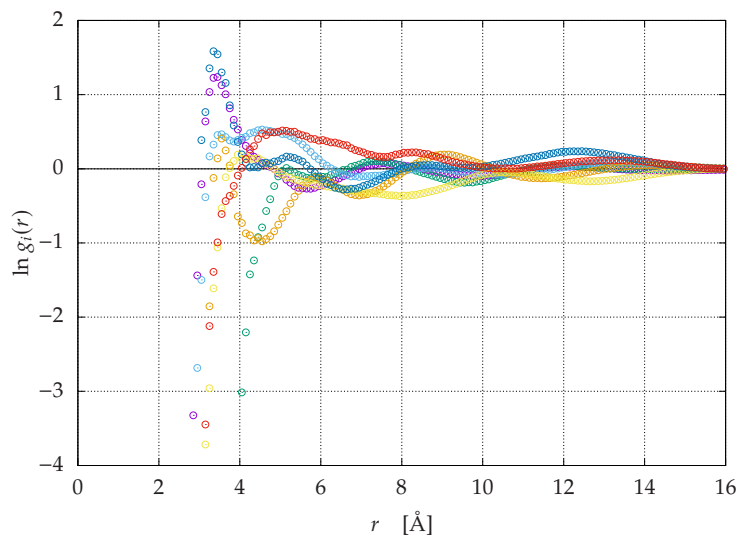
$$\begin{aligned} \frac{\partial(-\ln P)}{\partial \alpha_{a_j b_{ij}}} &= \sum_i^{\text{cells}} \left(1 - \frac{N_i}{\lambda_i}\right) \frac{\partial}{\partial \alpha_{a_j b_{ij}}} \lambda_i \\ &= \sum_i' \left(1 - \frac{N_i}{\lambda_i}\right) \lambda_i \\ &= \sum_i' (\lambda_i - N_i) \end{aligned} \quad (2.27)$$

where the prime (') on the sum denotes only cells that have a  $\lambda_i$  with an  $\alpha_{a_j b_{ij}}$  term, i.e. within the spherical shell of atom  $j$  at distance  $b_{ij}$ . This happens because the derivative of the exponential gives the function back if the variable is present in the exponential and is zero otherwise e.g.  $\frac{d}{dx} (e^{x+y}) = (x+y)' e^{x+y} = e^{x+y}$ . This is a non-linear equation in our fitting parameters  $\alpha$  because  $\lambda_i = e^{\alpha_{a_j b_{ij}}}$ . Unlike  $\chi^2$  this can not be solved analytically so we must take iterative steps to attempt to find the minimum. The second derivative of equation (2.25) is used to judge the direction and magnitude of each step along each parameter,

$$\frac{\partial^2(-\ln P)}{\partial \alpha_{a_j b_{ij}} \partial \alpha_{a_k b_{ik}}} = \sum_i'' \lambda_i \quad (2.28)$$

where the double prime (") is where the two spherical shells intersect.

A few resultant atomic distribution function fits from this procedure are shown in figure 2.16.



**Figure 2.16:** Fitted atomic distribution histograms of a few LO atom types. The distribution functions are fit and interpolated as the natural logarithm of  $g(r)$ .

## Chapter 3

# Elucidating Structural Evolution of Perylene Diimide Aggregates Using Vibrational Spectroscopy and Molecular Dynamics Simulations<sup>‡</sup>

### 3.1 Overview

Perylene diimides (PDIs) are a family of molecules that have potential applications to organic photovoltaics. These systems typically aggregate cofacially due to  $\pi$ -stacking interactions between the aromatic perylene cores. In this study, the structure and characteristics of aggregated N,N'-bis(2,6-diisopropylphenyl)-3,4,9,10-perylenetetracarboxylic diimide (common name lumogen orange), a perylene diimide (PDI) with sterically bulky imide functional groups, were investigated using both experimental vibrational spectroscopy and molecular dynamics (MD) simulations. Samples of lumogen orange dispersed in chloroform exhibited complex aggregation behavior, as evidenced by the evolution of the FTIR spectrum over a period of several hours. While for many PDI systems with less bulky imide functional groups aggregation is dominated by  $\pi$ -stacking interactions between perylene cores, MD simulations of lumogen orange dimers indicated a second, more energetically favorable aggregate structure mediated by "edge-to-edge" interactions between PDI units. Two-dimensional infrared spectroscopy together with orientational statistics obtained from MD simulations were employed to identify and rationalize aggregation-induced coupling between vibrational modes.

---

<sup>‡</sup>The molecular simulation, quantum mechanical calculations, and writing of this paper were done by Max Mattson. The experimental methods were performed by Thomas Green. This article was published with the following citation information: Mattson, M. A., Green, T. D., Lake, P. T., McCullagh, M., & Krummel, A. T. (2018). Elucidating structural evolution of perylene diimide aggregates using vibrational spectroscopy and molecular dynamics simulations. *The Journal of Physical Chemistry B*, 122(18), 4891-4900.

## 3.2 Introduction

Perylene diimides (PDIs) represent a class of organic molecules that have generated increasing interest for their use in solar energy applications.<sup>16–18</sup> They have been identified as potential alternatives to fullerene-based electron acceptors in bulk heterojunction solar cells due to their photostability, high electron mobility, and improved spectral overlap with the solar spectrum, allowing them to participate in solar light harvesting.<sup>19</sup> One key consideration in developing PDI systems is aggregation, which has a strong influence on the optical properties of these systems. The aggregation and self-assembly of PDI systems have been the subject of significant investigation.<sup>20,21</sup> Typically, the dominant driving force for aggregation in these systems is  $\pi$ - $\pi$  interactions between perylene cores, often leading to highly ordered column-like structures.<sup>22</sup> On one hand, thin films of PDI aggregates have been reported to exhibit very long exciton migration length scales ( $>2 \mu\text{m}$ ), suggesting that PDI aggregates should effectively transport excitons to charge separation sites; on the other hand, formation of these extended structures can have deleterious effects due to increased charge carrier trapping.<sup>23,24</sup>

A recent study of PDI thin films indicated that the morphology of polycrystalline PDI films plays an important role in the conductivity of the film and decreasing the structural order of the films had a positive effect on conductivity.<sup>25</sup> Reports detailing the influence of the microscopic structure on charge transport in PDI aggregate/conductive polymer blended films have also indicated that small disordered PDI aggregate domains are preferred due to reduced excimer-like relaxation.<sup>26</sup> Several studies have explored the influence of functionalization of PDI molecules on the resulting aggregates.<sup>27–30</sup> For example, sterically bulky functional groups attached at the imide position have

<sup>16</sup>Tang, C. W. *Applied Physics Letters* **1986**, *48*, 183–185.

<sup>17</sup>Zhang, X. et al. *Advanced Materials* **2013**, *25*, 5791–5797.

<sup>18</sup>Zhou, E. et al. *Angewandte Chemie International Edition* **2011**, *50*, 2799–2803.

<sup>19</sup>Nielsen, C. B. et al. *Accounts of Chemical Research* **2015**, *48*, 2803–2812.

<sup>20</sup>Zang, L.; Che, Y.; Moore, J. S. *Accounts of Chemical Research* **2008**, *41*, 1596–1608.

<sup>21</sup>Chen, S. et al. *Chemical Reviews* **2015**, *115*, 11967–11998.

<sup>22</sup>Würthner, F. et al. *Chemical Communications* **2006**, 1188–1190.

<sup>23</sup>Keivanidis, P. E.; Howard, I. A.; Friend, R. H. *Advanced Functional Materials* **2008**, *18*, 3189–3202.

<sup>24</sup>Mäkinen, A. J. et al. *Physical Review B* **1999**, *60*, 14683–14687.

<sup>25</sup>Russ, B. et al. *Advanced Materials* **2014**, *26*, 3473–3477.

<sup>26</sup>Ye, T. et al. *ACS Applied Materials & Interfaces* **2013**, *5*, 11844–11857.

<sup>27</sup>Balakrishnan, K. et al. *Journal of the American Chemical Society* **2006**, *128*, 7390–7398.

<sup>28</sup>Shao, C. et al. *Chemistry – A European Journal* **2012**, *18*, 13665–13677.

<sup>29</sup>Fennel, F. et al. *Journal of the American Chemical Society* **2013**, *135*, 18722–18725.

<sup>30</sup>Kaiser, T. E. et al. *Angewandte Chemie International Edition* **2007**, *46*, 5541–5544.

been employed to inhibit  $\pi$ -stack aggregate extension beyond dimerization by blocking additional molecules from interacting with the perylene core. In some cases, PDI functionalization has been reported to lead to complex aggregation behavior with more than one aggregate structure.<sup>28–30</sup> Another strategy for tuning intermolecular interactions employs so-called foldamers, or covalently linked PDI molecules with flexible linkers that allow the PDI units to aggregate but restrict the possible geometries available. A recent report identified one such system with two distinct aggregate structures in solution, with very different luminescent behavior.<sup>31</sup> Clearly, a detailed understanding of the structure and structural dynamics of solution phase PDI aggregates is a necessity in designing the next generation of high-efficiency PDI-based bulk heterojunction solar cells.

Infrared spectroscopy is a technique well suited to the investigation of aggregate structure due to the sensitivity of vibrational frequencies and oscillator strengths to changes in vibrational coupling and local environment or solvation. Two-dimensional infrared (2D IR) spectroscopy provides additional information through analysis of cross-peaks, which reflect coupling between vibrational modes. Structural information can be inferred then by combining experimental results with computational work. These approaches have been applied in the past to elucidate the structure of peptide oligomers,<sup>32</sup> proteins,<sup>33–35</sup> and DNA,<sup>36,37</sup> as well as metal-carbonyl structures<sup>38,39</sup> and aggregates of polyaromatic hydrocarbons.<sup>40</sup>

In this work, we employ a combination of linear IR and 2D IR spectroscopy together with molecular dynamics simulations to investigate the structure and structural evolution of aggregated samples of N,N'-bis(2,6-diisopropylphenyl)-3,4,9,10-perylenetetracarboxylic diimide (common name lumogen orange). We report evidence of the initial formation of  $\pi$ -stacked aggregates that over the course of several hours convert to extended aggregates. The slow-forming aggregate appears

<sup>28</sup>Shao, C. et al. *Chemistry – A European Journal* **2012**, *18*, 13665–13677.

<sup>29</sup>Fennel, F. et al. *Journal of the American Chemical Society* **2013**, *135*, 18722–18725.

<sup>30</sup>Kaiser, T. E. et al. *Angewandte Chemie International Edition* **2007**, *46*, 5541–5544.

<sup>31</sup>Samanta, S.; Chaudhuri, D. *The Journal of Physical Chemistry Letters* **2017**, *8*, 3427–3432.

<sup>32</sup>Zanni, M. T. et al. *The Journal of Physical Chemistry B* **2001**, *105*, 6520–6535.

<sup>33</sup>Wang, L. et al. *Journal of the American Chemical Society* **2011**, *133*, 16062–16071.

<sup>34</sup>Moran, S. D.; Zanni, M. T. *The Journal of Physical Chemistry Letters* **2014**, *5*, 1984–1993.

<sup>35</sup>Ganim, Z.; Tokmakoff, A. *Biophysical Journal* **2006**, *91*, 2636–2646.

<sup>36</sup>Krummel, A. T.; Mukherjee, P.; Zanni, M. T. *The Journal of Physical Chemistry B* **2003**, *107*, 9165–9169.

<sup>37</sup>Peng, C. S.; Jones, K. C.; Tokmakoff, A. *Journal of the American Chemical Society* **2011**, *133*, 15650–15660.

<sup>38</sup>Anna, J. M.; King, J. T.; Kubarych, K. J. *Inorganic Chemistry* **2011**, *50*, 9273–9283.

<sup>39</sup>Oudenhoven, T. A. et al. *The Journal of Chemical Physics* **2015**, *142*, 212449.

<sup>40</sup>Cyran, J. D.; Krummel, A. T. *The Journal of Chemical Physics* **2015**, *142*, 212435.

to form due to interactions between partially negative oxygen and partially positive hydrogen atoms on the molecules' periphery. Finally, we employ MD simulations of lumogen orange dimers and electronic structure calculations of selected dimer structures to extrapolate the structural and spectral features of the dimer to the experimental results acquired from a distribution of aggregate structures. These results lay the groundwork for investigations into the role of nuclear motions on charge transport in these aggregate systems; thus, these results will assist in the rational design of future PDI-based organic photovoltaics.

### 3.3 Experimental Methods

#### 3.3.1 Sample Preparation

Lumogen orange (N,N'-bis(2,6-diisopropylphenyl)-3,4,9,10-perylenetetracarboxylic diimide) was obtained from Tokyo Chemical Industry Co. and used as received. Chloroform was used as the solvent for all experiments. A 0.75 mM solution of lumogen orange in chloroform was prepared to investigate isolated molecules, and a 5 mM solution was prepared to investigate aggregates. All solutions were sonicated for 5 min before preparing samples for spectroscopic measurements.

#### 3.3.2 Linear IR and 2D IR Spectroscopy

Samples for spectroscopy were prepared by sandwiching an aliquot of the solution of interest between two CaF<sub>2</sub> windows along with a 250  $\mu\text{m}$  Teflon spacer to set the path length of the sample cell. FTIR spectra were recorded on a Vertex 70 FTIR spectrometer over the frequency range 1585–1740  $\text{cm}^{-1}$  with 1  $\text{cm}^{-1}$  resolution, in order to monitor the carbonyl and ring vibrational modes accessible for 2D IR experiments.

The 2D IR instrumentation employed here has been described in detail previously.<sup>41</sup> Briefly, the output of a regeneratively amplified Ti:sapphire femtosecond laser system (<50 fs, 2.6 W, centered at 785 nm) was delivered to an optical parametric amplifier fitted with a difference frequency generation stage to produce mid-IR pulses that were centered at 6  $\mu\text{m}$  with a 1200 nm full width at half-maximum (fwhm) and approximately 150 fs in duration. A ZnSe wedge split the output into a probe beam with 5% of the total pulse energy and a pump beam containing the remaining

---

<sup>41</sup>Cyran, J. D.; Nite, J. M.; Krummel, A. T. *The Journal of Physical Chemistry B* **2015**, *119*, Publisher: American Chemical Society, 8917–8925.

95% of the pulse energy. The pump beam was directed into a home-built acousto-optic mid-IR pulse shaper in order to generate phase-stable pulse pairs with a controlled delay,  $\tau$ . The delay was scanned up to a 2500 fs maximum delay with 25 fs steps. Group velocity and third-order dispersion corrections were written into the acoustic masks to correct for dispersion resulting from transmission through the germanium acousto-optic modulator as well as other transmissive optics. A four-frame phase-cycling scheme and a partially rotating frame ( $1400\text{ cm}^{-1}$ ) were applied via the pulse shaper in order to eliminate the transient absorption signal and pump scatter in the final 2D IR spectra. The resulting colinear pulse pairs were focused into the sample. The probe line passed through a computer-controlled variable delay line to control the delay time between the second and third laser pulses,  $T_w$ , before being focused into the sample. The pump pulses and probe pulse were overlapped spatially and temporally in the focal plane of the sample. The probe was recollimated and steered to a monochromator and 64-element mercury cadmium telluride (MCT) array detector. The probe frequency axis,  $\omega_{\text{probe}}$ , is generated from the array detector, and the pump frequency axis,  $\omega_{\text{pump}}$ , is generated by taking the Fourier transform of the signal at each pixel on the array detector with respect to  $\tau$ . Taking into account the experimental geometry, optical components, optical layout, and time delays employed, the resulting spectra have  $13\text{ cm}^{-1}$  spectral resolution on the pump axis and  $2.0\text{ cm}^{-1}$  spectral resolution along the probe axis.

## 3.4 Computational Methods

### 3.4.1 Molecular Dynamics

Enhanced sampling molecular dynamics (MD) simulations were performed using replica exchange umbrella sampling (REUS) in the NAMD molecular dynamics software package<sup>42</sup> to sample the potential energy surface of dimerization between two lumogen orange molecules in chloroform as a function of the center-of-mass distance between the two molecules. Lumogen orange was parametrized using GAFF,<sup>15</sup> and charges were found with RESP;<sup>43</sup> the AMBER chloroform box was used for the solvent.<sup>44</sup> Harmonic biasing potentials were placed  $0.5\text{ \AA}$  apart along the collective variable axis and were given spring constants of  $20.0\text{ kcal mol}^{-1}\text{ \AA}^{-2}$ . These parameters were used

<sup>42</sup>Phillips, J. C. et al. *Journal of Computational Chemistry* **2005**, *26*, 1781–1802.

<sup>15</sup>Wang, J. et al. *Journal of Computational Chemistry* **2004**, *25*, 1157–1174.

<sup>43</sup>Bayly, C. I. et al. *The Journal of Physical Chemistry* **1993**, *97*, 10269–10280.

<sup>44</sup>Fox, T.; Kollman, P. A. *The Journal of Physical Chemistry B* **1998**, *102*, 8070–8079.

because they struck a nice balance of ensuring sufficient sampling overlap between neighboring windows, while maintaining a strong enough bias to ensure sampling of barrier regions, as well as an exchange probability between neighboring windows of  $\sim 10\%$ . The resultant center-of-mass distances were analyzed using the weighted histogram analysis method (WHAM)<sup>45</sup> to build the potential of mean force (PMF). The PMF was constructed using a home-built binning script to check for nonergodicity among the biased windows. Each window in the REUS simulations was run for 100 ns in the NVT ensemble with a total of 25,000 exchange attempts per window with its neighbors.

### 3.4.2 Quantum Calculations

Vibrational frequency calculations were performed using Gaussian 09<sup>46</sup> for the lumogen orange monomer and dimer structures that were representative of the most common structures in each stable dimer configuration. The APFD functional was used for its dispersive properties to stabilize  $\pi$ -stacking interactions between the two monomers. During the geometry optimization, the  $\pi$ -stacked dimer stabilized near its starting orientation taken from REUS trajectories due to the dispersive element of the APFD functional. However, the 11 Å dimer would change orientations drastically from the starting point to the geometry-optimized structure. To keep the starting structures intact, which were taken from MD simulations, the two meta-carbons on the R-groups of each molecule were frozen in their starting positions for the geometry optimization; this protocol will lead to imaginary frequencies, because these carbons were not in minima of the potential energy surface. To handle this issue, each frozen meta-carbon was given a mass of 1000 AMU to effectively remove their contribution from the vibrational frequency calculation. As a result, the 11 Å dimers were effectively frozen in their starting configurations, able to rotate or translate minimally relative to one another. The protocol described here is essential because the relative orientations of the monomers in the dimer configuration govern the strength and nature of vibrational coupling between the monomer units. Without instituting this position constraining procedure, the 11 Å dimers change relative orientations and positions significantly during the geometry optimization.

---

<sup>45</sup>Kumar, S. et al. *Journal of Computational Chemistry* **1992**, *13*, 1011–1021.

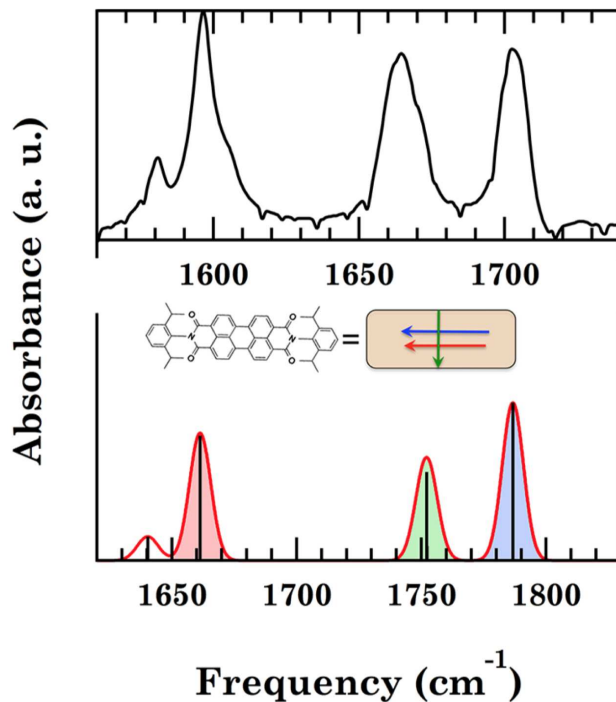
<sup>46</sup>Frisch, M. et al. *Gaussian Inc, Wallingford CT, 2013* **2013**.



### 3.5 Results and Discussion

The experimental FTIR spectrum of 0.75 mM lumogen orange dissolved in chloroform contained three peaks in the frequency range 1570–1780  $\text{cm}^{-1}$ . This sample was taken to represent isolated (i.e., unaggregated) PDI, as the concentration was less than the critical aggregation concentration (approximately 2 mM); this was further evidenced by UV-visible spectroscopy of the samples with 0.75 and 5.0 mM concentration in lumogen orange, respectively (see the Supporting Information, Figure 3.6). Electronic structure calculations were employed to assign the observed peaks in the spectrum (Figure 3.1). There are four calculated carbonyl vibrational modes in PDI; however, two are dark modes possessing much lower oscillator strengths than the others. The high energy carbonyl mode at 1786  $\text{cm}^{-1}$  consists of symmetric carbonyl stretches on each imide group that are antisymmetric with the other imide group. This mode is paired with a dark carbonyl mode at 1791  $\text{cm}^{-1}$  where all carbonyls are symmetric with each other. The lower energy carbonyl mode at 1752  $\text{cm}^{-1}$  consists of carbonyl stretches predominantly on one imide that are antisymmetric from each other but symmetric with the other imide. Finally, there is a lower intensity mode at 1753  $\text{cm}^{-1}$  that shares the same symmetry as the 1752  $\text{cm}^{-1}$  mode but with more of the oscillator strength on the other imide. Using the calculated spectrum, the FTIR peaks centered at 1705 and 1667  $\text{cm}^{-1}$  were assigned as carbonyl stretching modes with transition dipoles oriented along the major and minor molecular axes, respectively. The peak at 1596  $\text{cm}^{-1}$  was attributed to the ring motions of the perylene core, with its transition dipole oriented along the major molecular axis (Figure 3.1, inset). It should be noted that the lower intensity peak at approximately 1580  $\text{cm}^{-1}$  was attributed to a ring mode; however, it is not included in the analysis presented here.

At increased concentration (5 mM), the profile of the FTIR spectrum was observed to evolve over time. Spectra were collected at 15 min intervals over the course of several hours to investigate the molecular origin of this temporal evolution (Figure 3.2A). Initially, the FTIR spectrum resembled that of the monomer solution (monomer FTIR spectrum scaled for concentration represented by the dashed line in Figure 3.2A), with three peaks approximately equal in intensity. As time progressed, the high-energy carbonyl peak dramatically increased in intensity and concomitantly shifted to higher frequency. These results suggest the formation of a new type of aggregate structure. Further visual inspection of the spectra collected during the first 1.5 h revealed that the intensity growth



**Figure 3.1:** Experimental FTIR spectrum of 0.75 mM in chloroform (top) and calculated vibrational spectrum of isolated lumogen orange using the APFD functional (bottom). The frequency axis of the calculated spectrum is unscaled. The inset depicts the PDI molecule represented as a rectangle with the transition dipoles of the relevant vibrational modes depicted as color-coded arrows (arbitrary length), which represent the vibrational modes responsible for the corresponding shaded IR peaks.

was asymmetric, favoring the high-energy side of the peak (see Figure 3.2A, inset). This was a clear indication that the FTIR peak was comprised of multiple convoluted peaks. In order to describe these dynamics with mode-specificity, each FTIR spectrum was fit to Lorentzian peaks, the frequency and area of which were then analyzed individually.

**Table 3.1:** Summary of FTIR fit results.

peak	<u>t=0 min</u>			<u>t=360 min</u>		
	freq (cm <sup>-1</sup> )	fwhm (cm <sup>-1</sup> )	area (a.u.)	freq (cm <sup>-1</sup> )	fwhm (cm <sup>-1</sup> )	area (a.u.)
1	1596	8	12	1596	7	11
2	1606	6	1	1606	8	2
3	1662	10	10	1663	12	17
4	1668	11	9	1671	8	7
5	1701	7	7	1706	11	31
6	1706	8	9	1711	12	87

Parts B and C of Figure 3.2 depict the best fit of the initial and final FTIR spectra. Fit results are summarized in Table 1. In this analysis, we focus specifically on the two peaks—peak 5 and peak 6—comprising the high-energy carbonyl peak in the FTIR spectra. Both of these peaks exhibited similar trends in their frequency shift and peak areas as a function of time. Initially, at  $t = 0$  min, the center frequency of peak 6 was determined to be  $1705.9\text{ cm}^{-1}$  (Figure 3.2E). After approximately 45 min, the peak began to shift to higher frequency, shifting to approximately  $1710\text{ cm}^{-1}$  after 120 min before beginning to plateau. At  $t = 360$  min, the peak was centered at  $1711.3\text{ cm}^{-1}$ . The area of peak 6 was observed to increase approximately linearly over the entire time scale investigated. Peak 5 was initially centered at  $1700.8\text{ cm}^{-1}$  and shifted to  $1705.7\text{ cm}^{-1}$  over the course of the experiment. Although the frequency shift of peak 5 also began to plateau, it was less abrupt than peak 6. The area of this peak was also observed to increase linearly over the investigated time range.

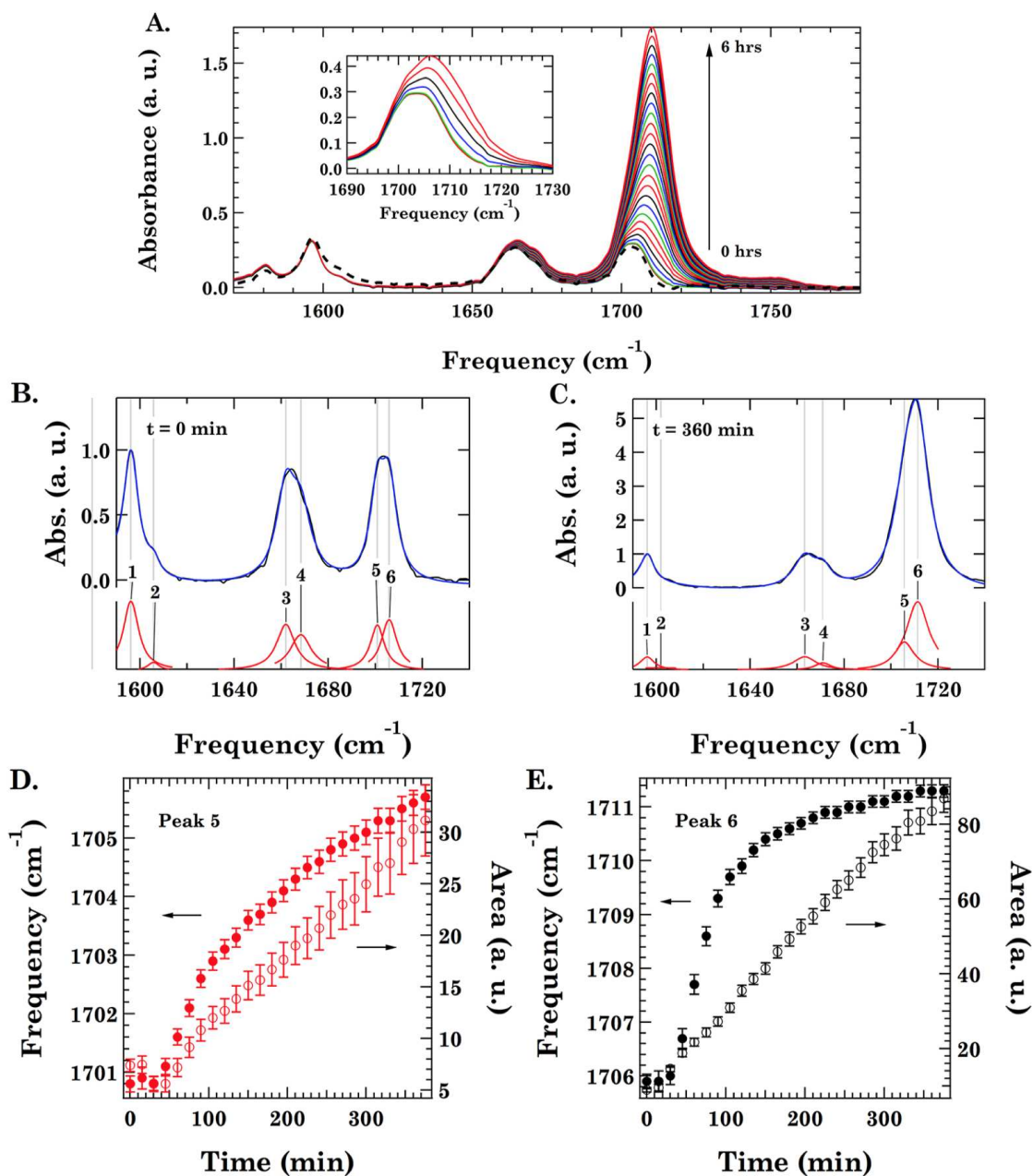
These results were interpreted to reflect a change in aggregate structure, leading to a shift in the vibrational frequencies, accompanied by extended aggregate growth, which influences oscillator strength and therefore peak area. MD simulations were employed to further elucidate the possible dimer unit cell structures that contribute to the aggregates that were formed.

To investigate the likely structural conformations that contribute to the aggregate structures, MD simulations were used to construct the PMF (Figure 3.3A) as a function of the center-of-mass distance between two PDI cores. The PMF suggests that there are two stable center-of-mass distances where the PDIs are interacting. The first is the well at  $\sim 3.6\text{ \AA}$  which is synonymous with a  $\pi$ -stacking interaction, as shown in the inset diagram. The second is the broad well at  $\sim 11.4\text{ \AA}$  which consists of dimer structures where the edge of one PDI core is interacting with the other PDI molecule. Many of the structures from this "edge-on" well are stabilized by an interaction between the carbonyl oxygen of one PDI and a bay-position hydrogen of the other PDI, reminiscent of the nonclassical hydrogen bonds observed in naphthalenediimide self-assemblies.<sup>47,48</sup> The peak of the barrier at  $\sim 5.5\text{ \AA}$  between the  $\pi$ -stack minimum and the edge-on minimum is an effect of the solvent (see Figure 3.7 in the Supporting Information). This center-of-mass separation corresponds

---

<sup>47</sup>Ponnuswamy, N. et al. In *Constitutional Dynamic Chemistry*, Barboiu, M., Ed.; Topics in Current Chemistry; Springer: Berlin, Heidelberg, 2012, pp 217–260.

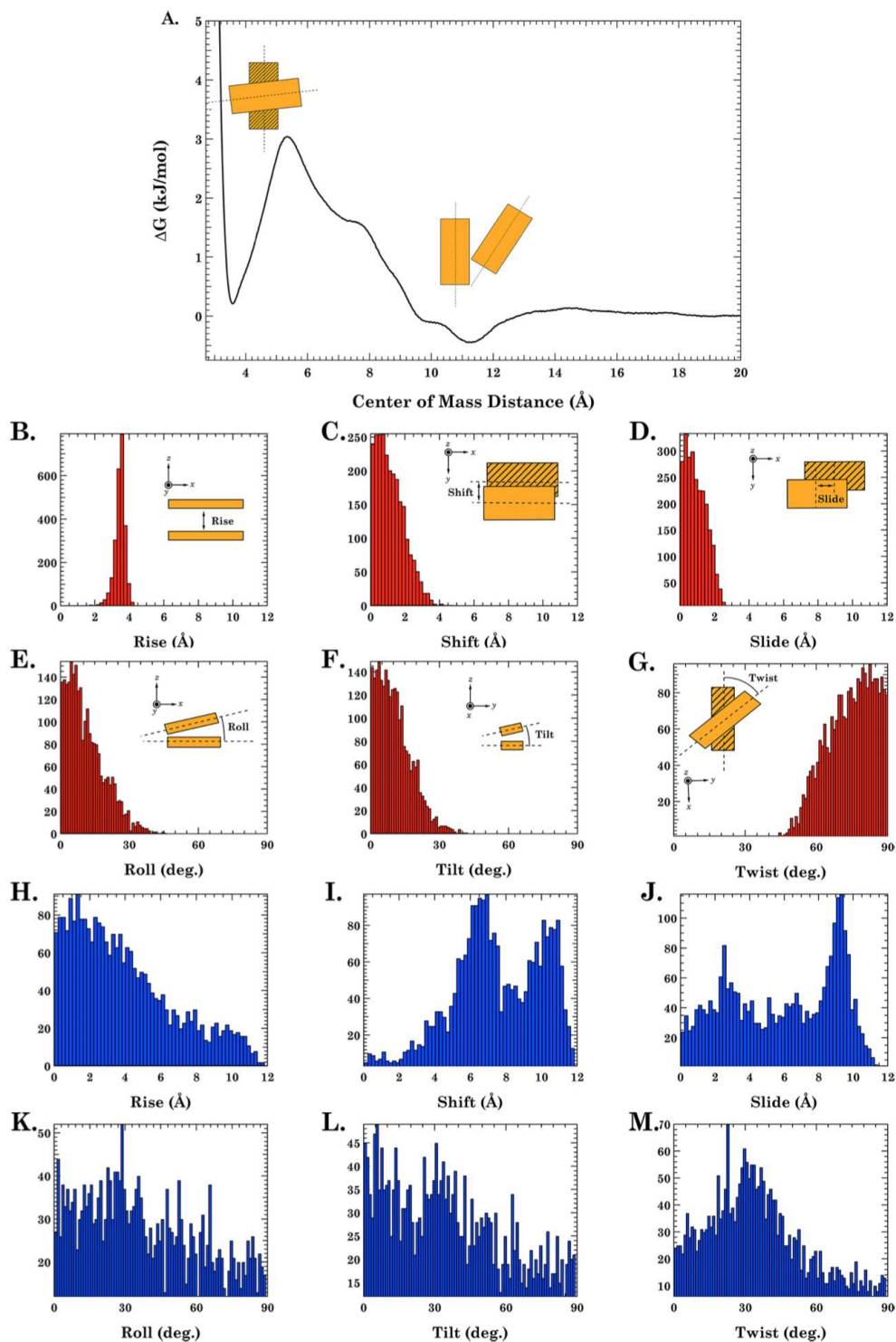
<sup>48</sup>Tambara, K. et al. *Organic & Biomolecular Chemistry* **2014**, *12*, 607–614.



**Figure 3.2:** (A) Temporal evolution of the FTIR spectrum of 5 mM lumogen orange. Spectra were collected at 15 min intervals for 6 h total; the dashed line reflects the monomer FTIR spectrum scaled for concentration. (B and C) The best fits of the first and last spectra in the series, respectively. (D and E) Frequency (solid circles) and area (open circles) of peak 5 and peak 6 from the fits, respectively, as a function of time.

to structures where the faces of the PDI cores are too far away to  $\pi$ -stack but too close to participate in the edge-on nonclassical hydrogen bonding.

There are two aspects of Figure 3.3A that are surprising: edge-on is the most energetically stable center-of-mass separation, and there is a large barrier to  $\pi$ -stack, which is not typically the



**Figure 3.3:** (A) PMF as a function of center-of-mass distances during dimerization of lumogen orange. (B-G) (red) Orientational statistics from the 3.6  $\text{\AA}$  well. (H-M) (blue) Orientational statistics from the 11  $\text{\AA}$  well.

case with molecules that share the PDI core. In relation to what leads to this PMF result, we have conducted MD simulations to calculate the PMFs of PDI cores with a range of R-groups (see Figure 3.8 in the Supporting Information). As the chain-like R-groups become more rigid, the PMFs form more stable interactions at larger center-of-mass distances. However, as the R-group becomes more sterically bulky and less able to lie flat in the plane of the core, there is a decreased stability at distances between 5 and 10 Å, where the cores are too close to have the edge-on interaction but before the PDI cores are close enough to  $\pi$ -stack. Study of the aggregation of PDIs is usually in relation to its propensity to  $\pi$ -stack;<sup>27,28,49</sup> however, the PMF resultant from the REUS MD simulations for lumogen orange shows that there are two favorable center-of-mass separation distances for the PDI dimer:  $\sim 3.6$  and  $\sim 11$  Å separation. The 3.6 Å separation distance corresponds to a cofacially  $\pi$ -stacked dimer. The 11 Å separation distance corresponds to an "edge-on" aggregate where the edge of one PDI is interacting with the other PDI (see the Figure 3.3A insets for representative structures) which, as mentioned previously, is the most energetically favorable position in the PMF. This creates a unique situation among the PDIs tested here of having two distinct dimer conformations.

MD simulations yielded orientational statistics describing the distribution of structures corresponding to each well in the PMF. The orientations of the PDI molecules relative to one another can be uniquely described with a total of six parameters: three distance and three angular. Histograms of each orientational parameter are plotted in Figure 3.3 with a corresponding inset depiction of each parameter. The distance parameters, slide, shift, and rise, describe the translation of one PDI along the long axis, short axis, and axis perpendicular to the perylene face, respectively. The angular parameters, roll, tilt, and twist, describe the rotation about the long, short, and perylene-face-normal axes, respectively.

The relative orientation of the  $\pi$ -stacked dimer (Figure 3.3B-G) is narrowly distributed in the MD simulations, whereas the 11 Å dimer is not. All three distance parameters for the  $\pi$ -stack dimer (Figure 3.3B-D) have standard deviations  $< 2$  Å. The angular parameters (Figure 3.3E,F) have standard deviations  $< 20^\circ$  for roll and tilt. Twist has a larger standard deviation because the flat cores of the PDIs can stay  $\pi$ -stacked with cofacial contact and twist significantly before the sterically

<sup>27</sup>Balakrishnan, K. et al. *Journal of the American Chemical Society* **2006**, *128*, 7390–7398.

<sup>28</sup>Shao, C. et al. *Chemistry – A European Journal* **2012**, *18*, 13665–13677.

<sup>49</sup>Sukul, P. K. et al. *Chemical Communications* **2011**, *47*, 11858–11860.

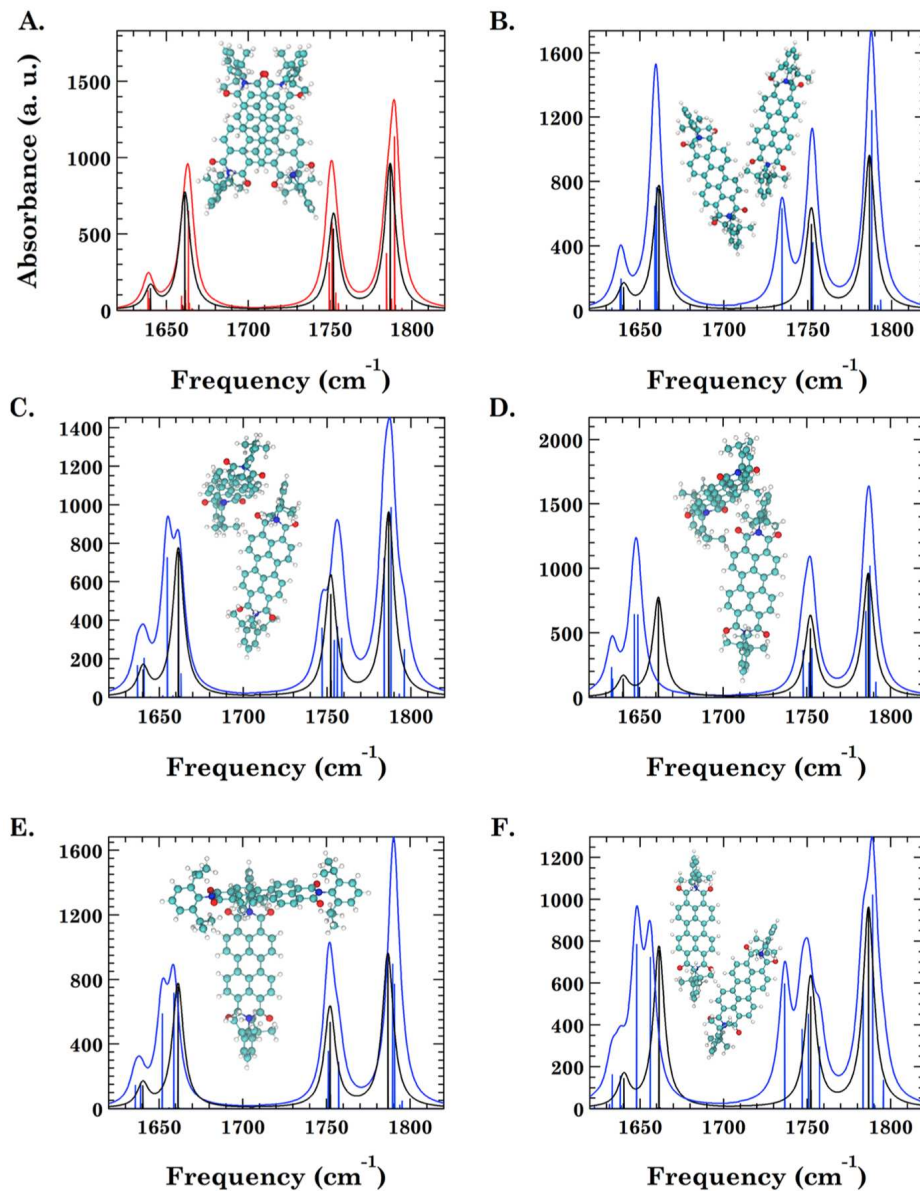
hindering R-groups obstruct further rotation. This simplifies the assignment of the vibrational spectra for the  $\pi$ -stacked dimer because the ensemble of significantly contributing structures is small. However, in the 11 Å dimer, there are a broad range of possible structures because each orientational parameter is at least moderately disperse. This complicates matters spectroscopically because the ensemble of potential significantly contributing structures is large. This dispersity may occur in part because the PMF was calculated for a dimer only and the addition of more lumogen orange molecules could facilitate a more stable and narrowly distributed orientational distribution for the monomer units with respect to one another.

The results of the MD simulations suggest three dominant structures in this system: cofacial aggregates mediated by  $\pi$ - $\pi$  interactions between the conjugated perylene diimide cores, an "edge-on" aggregate likely formed through interactions between the partially negatively charged oxygen atoms and weakly positively charged hydrogen atoms in the bay positions of neighboring PDI molecules, and isolated PDI monomers. Equilibrium constants were calculated from the REUS dimer PMF and found to be 77.<sup>42</sup> for the reaction of  $\pi$ -dimers to edge-on dimers and  $6.51 \times 10^{-5} \text{ \AA}^{-3}$  for the reaction from the edge-on dimer to two PDI monomers. In order to guide the assignment of experimental spectral features, electronic structure calculations were performed for dimer structures from each minimum on the PMF surface. Because the 11 Å minimum in the PMF is broad, implying that many structures are energetically very similar, spectra of several structures within this minimum were calculated.

The calculated IR stick spectra of isolated lumogen orange and a  $\pi$ -stack dimer, convolved with Lorentzian functions with an  $8 \text{ cm}^{-1}$  line width, are presented in Figure 3.4A. The  $\pi$ -stack dimer spectrum is very similar to the isolated PDI spectrum. However, whereas each peak in the isolated PDI is comprised of only one mode with significant oscillator strength, the peaks in the  $\pi$ -stack spectrum are comprised of several modes close in frequency with significant oscillator strength. The  $\pi$ -stack modes are comprised of similar atomic motions compared with the monomer calculation, but they are delocalized over both molecules. For example, the high energy carbonyl mode in the  $\pi$ -stack dimer has the same symmetric carbonyl stretch as the monomer high energy carbonyl mode; however, the atomic motion is delocalized across both molecules. Despite these differences, the relative intensities of the peaks and their peak widths are very similar between

---

<sup>42</sup>Phillips, J. C. et al. *Journal of Computational Chemistry* **2005**, 26, 1781–1802.



**Figure 3.4:** (A) Calculated IR spectrum of a  $\pi$ -stacked dimer (red) compared to the calculated spectrum of the isolated species (black). (B-F) Calculated IR spectra of representative structures from the 11 Å well (blue) compared to the calculated spectrum of isolated lumogen orange (black).

these two spectra, which makes assignment of experimental spectra to one of these two species ambiguous. We will discuss differentiating between these two species using 2D IR spectroscopy below. Parts B-F of Figure 3.4 depict the calculated spectra of five different "edge-on" dimer structures extracted from the 11 Å minimum in the PMF. Similar to the  $\pi$ -stacked dimers, each peak is comprised of multiple modes. Comparing the spectra, it is evident that, while these five structures produce unique spectra, there are some general trends that appear to be common across



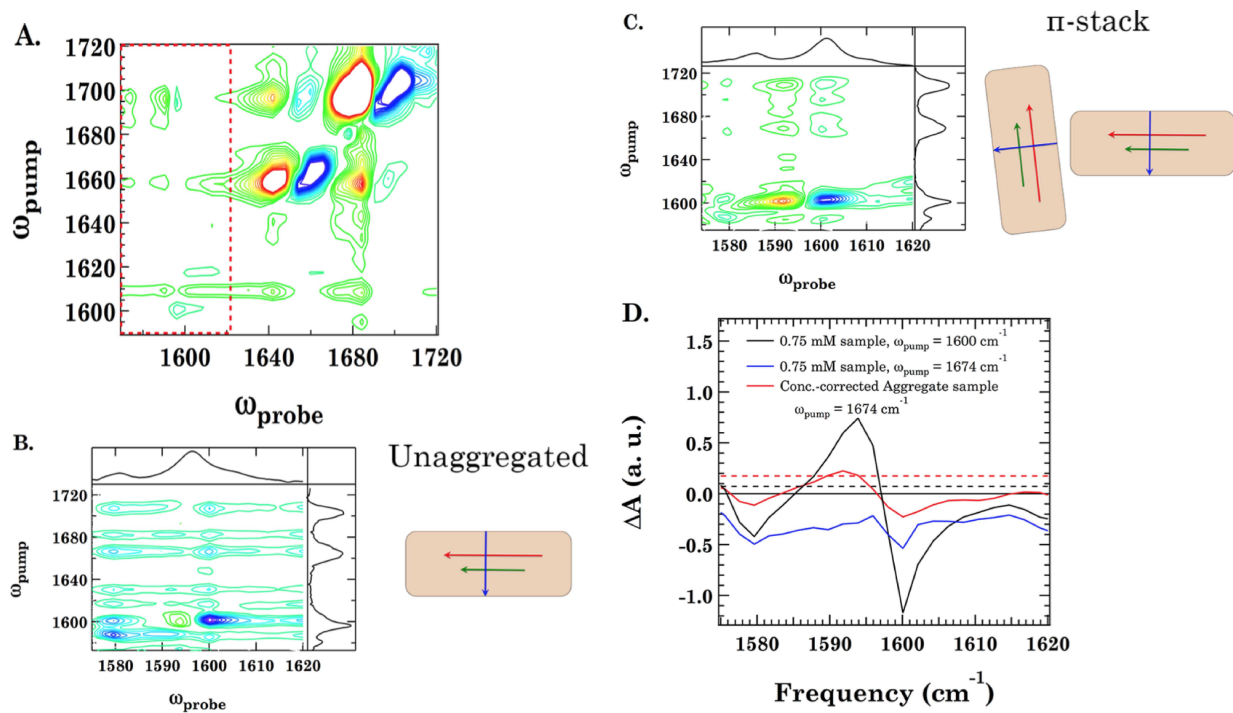
each of the spectra. In general, the intensity of the high-energy carbonyl mode relative to the ring mode tends to be increased for the "edge-on" dimers as compared with the isolated PDI spectra. The lower energy carbonyl peak also tends to be broadened in the dimer spectra. These general trends are in reasonable agreement with the experimental FTIR spectra collected toward the end of the time-lapse experiments. In these experimental data, the high-energy carbonyl peak grows in significantly and shifts to higher frequency by approximately  $5\text{ cm}^{-1}$ , and the low energy carbonyl mode broadens slightly, with the fwhm increasing from approximately  $15$  to  $20\text{ cm}^{-1}$ . On the basis of these trends in the simulated spectra, taken together with the molecular dynamics results, we assign the slow evolution of the experimental FTIR spectrum to reflect the formation of "edge-on" interactions within the growing aggregates. This assignment is also supported by the observation that the experimental peak areas grow significantly, indicating the formation of extended aggregates. Such extended growth of  $\pi$ -stacked aggregates is unlikely due to steric hindrance from the functional groups attached to the nitrogen atoms of the PDI.

It should be noted that there is a discrepancy between the ring mode in the simulated and experimental spectra. While the ring mode in several of the simulated dimer spectra was shifted to lower frequency compared to the isolated PDI spectrum, there was no observed shift in the ring mode in the experimental data. However, it is likely that the experimental spectra represent an ensemble of several aggregate structures corresponding to the  $11\text{ \AA}$  minimum from the PMF, considering the PDI spectrum continues to evolve after 6 h. It is also likely that, as small aggregates extend, the range of possible structures becomes narrowed.

While linear spectra were sufficient to assign the formation of "edge-on" aggregates, the simulated spectra of the isolated PDI and  $\pi$ -stacked dimer were too similar to unambiguously assign experimental spectra. Use of 2D IR spectroscopy, however, did allow differentiation between these species. Parts A and B of Figure 3.5 present 2D IR spectra of isolated lumogen orange and aggregated lumogen orange, respectively. The probe axis is expanded to focus on the ring mode region. In both the isolated PDI spectrum and the aggregate spectrum, the ground state bleach and anharmonically shifted excited state emission peaks corresponding to the  $v = 0 \rightarrow 1$  and  $v = 1 \rightarrow 2$  transitions of the ring mode are clearly visible at  $\omega_{\text{pump}} = 1600\text{ cm}^{-1}$  and  $\omega_{\text{probe}} = 1600$  and  $1594\text{ cm}^{-1}$ , respectively. If the ring mode was coupled to other vibrational modes, additional peak pairs would be evident located at the probe frequency of the ring mode and at the frequency

of the other mode along the pump axis. For example, if the ring mode was coupled to the lower energy carbonyl mode, cross-peaks would appear in the spectrum at  $\omega_{\text{pump}} = 1667 \text{ cm}^{-1}$  and  $\omega_{\text{probe}} = 1600$  and  $1594 \text{ cm}^{-1}$ . Although there is some noise in the spectrum of the isolated PDI due to the lower concentration, there is no peak pair at the frequency of the carbonyl mode, indicating a lack of coupling between these two modes. This was expected on the basis of the electronic structure calculations, which indicated that the transition dipoles of the low energy carbonyl mode and the ring mode are nearly orthogonal. In the 2DIR spectrum of the aggregate, however, there is a pair of cross-peaks between the ring mode and the low energy carbonyl mode. This can be rationalized by considering the orientational statistics from the MD simulations. Due to the steric hindrance from the imide functional groups, the  $\pi$ -stack aggregate forms with an almost  $90^\circ$  twist between individual PDI units. One consequence of this twist angle, together with the very small distance between  $\pi$ -stacked PDI molecules ( $<4 \text{ \AA}$ ), is that the transition dipole of the ring mode of one PDI unit in the dimer becomes aligned nearly parallel (antiparallel) with the transition dipole of the carbonyl of its neighbor, thus facilitating intermolecular coupling. The presence or absence of this cross-peak, then, can be used to differentiate between the isolated PDI and the  $\pi$ -stack aggregate.

In order to reliably assign the molecular species based on this peak in the 2D IR spectrum, it was necessary to rule out the possibility that the cross-peak is present in the low concentration sample but obscured by the noise. In other words, is it possible that the cross-peak is present regardless of aggregation state and it becomes apparent in the high concentration spectrum simply due to typical Beer's law concentration dependence? We addressed this question in the following manner: the intensity of the high concentration spectrum was corrected for concentration, assuming a linear relationship between absorbance and concentration. This concentration-corrected intensity could then be compared to the level of noise in the low concentration spectrum. If the corrected signal intensity rises above two standard deviations of the noise level in the low-concentration spectrum, then the cross-peak would be visible if it was present in the low concentration spectrum. Figure 3.5C depicts slices of the 2D IR spectra at the pump frequency corresponding to the center of the cross-peak and the diagonal peak of the ring mode. The noise level was quantified by calculating the average and standard deviation of the signal of the frequency range of the ring mode  $\nu = 0 \rightarrow 1$  and  $\nu = 1 \rightarrow 2$  transitions on the array axis at pump frequencies that are not resonant with any other transitions. This average plus one and two standard deviations is represented by the dashed



**Figure 3.5:** (A) 2DIR spectrum of 5 mM lumogen orange in  $\text{CHCl}_3$ . The red box highlights the region containing the cross-peaks reporting on coupling between carbonyl and ring vibrational modes. Subsequent 2DIR spectra were collected with higher resolution in this region. (B) 2DIR spectra of 0.75 mM lumogen orange in chloroform and the model of unaggregated lumogen orange depicting transition dipoles. (C) 2DIR spectrum of 5 mM lumogen orange scaled for concentration assuming Beer's law, in order to allow direct comparison of signals between the two samples. The model depicts a  $\pi$ -stacked dimer (shifted horizontally for clarity) demonstrating that the transition dipole of the ring mode (red arrow) becomes almost parallel with respect to the transition dipole of the low energy carbonyl mode (green arrow) of its neighbor due to the large twist angle between molecules. (D) Slices taken from the 2DIR spectra taken at pump frequencies corresponding to the low-energy carbonyl peak. Dashed lines indicate one and two standard deviations of the noise above the average level in the 0.75 mM spectrum, which were utilized to determine cross-peaks present in the spectra.

black and red lines, respectively. The concentration-corrected intensity exceeds both of these levels, suggesting that it is very unlikely that the cross-peak is the result of concentration alone. New coupling due to the aggregate structure contributes to the appearance of the cross-peak in the high concentration 2DIR spectrum.

## 3.6 Conclusion

Linear and 2D IR spectroscopy together with MD simulations were employed to probe the structure and temporal evolution of PDI aggregates. A new aggregation state, other than the typical  $\pi$ -stack, has been discovered, resulting in three dominant aggregation states for lumogen orange in chloroform, namely,  $\pi$ -stacked dimers and "edge-on" dimers as well as unaggregated lumogen orange. IR spectra of each of these species were calculated, and comparison of calculated spectra to experimental FTIR data revealed the formation of the "edge-on" aggregate over the course of several hours. Although linear FTIR spectroscopy was insufficient to distinguish between  $\pi$ -stacked aggregates and unaggregated lumogen orange, 2D IR measurements revealed aggregation-induced coupling between the carbonyl stretching and ring vibrational modes. Additional work is currently underway to investigate the vibrational coupling and dynamics of these aggregate structures as well as investigate the new edge-on state.

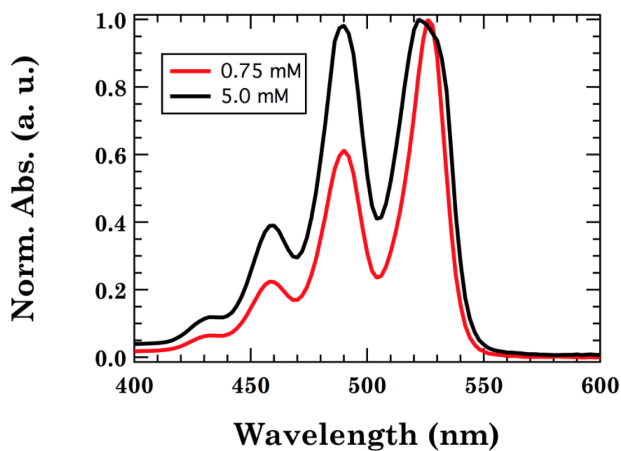
## 3.7 Supporting Information

### 3.7.1 Computational Methods

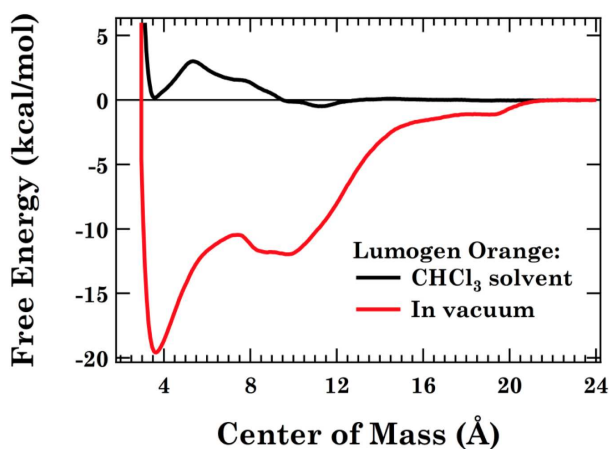
#### 3.7.1.1 Molecular Dynamics

Enhanced sampling Molecular Dynamics (MD) simulations were performed using Umbrella Sampling (US), and Replica Exchange Umbrella Sampling (REUS) in the AMBER and NAMD molecular dynamics software packages respectively. All simulations were of the Perylene Diimide (PDI) derivatives dissolved in chloroform with the center-of-mass distance between PDIs as the collective variable. Umbrella Sampling simulations were performed first on each PDI derivative. If there were no apparent issues with sampling ergodicity then the potential of mean force, (PMF), calculated using the US results. For the PDI derivatives that showed ergodicity issues REUS simulations were performed to enhance the sampling across simulation windows. All PDI derivatives

were parameterized using in the same methods described in the Computational Methods section of this paper.



**Figure 3.6:** UV-VIS spectra of 0.75mM (red) and 5.0 mM (black) lumogen orange in  $\text{CHCl}_3$ , representing unaggregated and aggregated samples, respectively.



**Figure 3.7:** PMF of lumogen orange dimer simulated in  $\text{CHCl}_3$  and in vacuum.

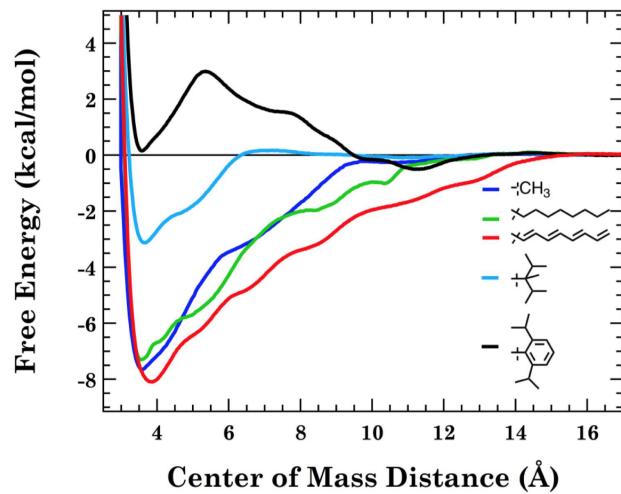


Figure 3.8: Influence of imide functional group on calculated dimer PMF.

## Chapter 4

### Vibrational Properties of Solvent Dependent

### Violanthrone-79 Aggregates Using Two-Dimensional

### Infrared Spectroscopy and Molecular Dynamics

### Simulations<sup>‡</sup>

#### 4.1 Overview

Violanthrone based compounds have received attention for their strong absorption in the visible spectral range as well as their charge transport properties and the application of these characteristics for materials sciences. Violanthrone derivatives are also known for their propensity to aggregate via  $\pi$ - $\pi$  stacking, which affects their absorption and charge transport properties. Therefore, it is of critical importance to understand what factors influence these  $\pi$ -stacking interactions. Violanthrone-79 samples in chloroform and tetrahydrofuran were studied for their notable spectral differences in the carbonyl region which suggest differences in the self-assembled structures. FTIR spectra were compared to quantum frequency calculations from structures generated using molecular dynamics simulations showing that tetrahydrofuran supports a larger ratio of parallel  $\pi$ -stacked aggregates while chloroform supports antiparallel  $\pi$ -stacked aggregates. Furthermore, two-dimensional infrared spectroscopy showed differences in the rate of vibrational energy transfer between these two systems, suggesting differences in the vibrational energy delocalization between the two aggregate structures which was corroborated by quantum mechanical vibrational frequency calculations.

#### 4.2 Introduction

Understanding molecular aggregation driven by  $\pi$ -stacking is critical for understanding how materials processing may impact the utility of organic dyes in photoelectrochemical cells.  $\pi$ - $\pi$

---

<sup>‡</sup>This manuscript is currently in preparation for submission. The molecular simulation and quantum calculations in this paper as well as much of the writing was performed by Max Mattson. The experimental methods and initial writing were done by Chris Kuhs.

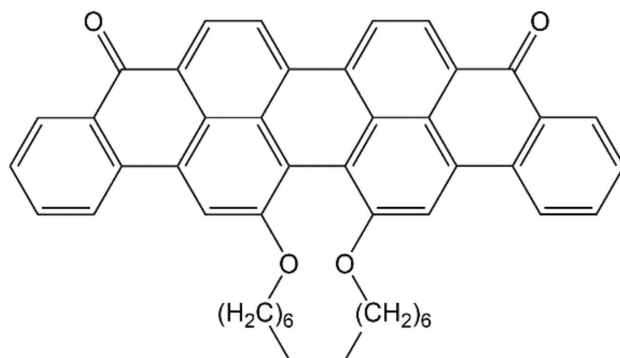
interactions between aromatic molecules are an important driving force for the self-assembly of DNA,<sup>50,51</sup> protein tertiary structure,<sup>52,53</sup> and polycyclic aromatic hydrocarbons (PAH)<sup>5,40,41</sup> such as violanthrone and perylene diimide derivatives. Polycyclic aromatic hydrocarbons are organic species with fused polyaromatic cores that are of interest because of their varied uses in the materials sciences.

Functionalized PAHs have led to technological advancements in solar cells,<sup>54</sup> laser dyes,<sup>55</sup> and liquid crystals.<sup>56</sup> However, tailoring functional groups on the PAH core is only one avenue through which the aggregation characteristics can be manipulated. The choice of solvent is also crucial in determining the structural characteristics of PAH aggregates. Consequently, the effect of solvent on  $\pi$ - $\pi$  interactions has been the topic of several studies.<sup>57-59</sup> Polar solvents will promote  $\pi$ - $\pi$  interaction as explained by solvophobic theory. However, the effect of solvent polarity on aggregate structure and energy delocalization therein, concepts that are critical in photoelectrochemical cells, is still unclear. Molecular orientation within aggregates and its impact on energy delocalization can be explored by employing experimental techniques that examine vibrational characteristics in PAH aggregates. In doing so, the effect of solvent on structural and energy delocalization properties can be better understood.

One PAH that has received attention over the years for its charge transport and absorptive properties is violanthrone. Violanthrone is a PAH that has been used as an analog to asphaltenes, but it has also shown promise as an organic dye for solar cell technology due to its large aromatic core making it ideal for absorbing solar radiation. Recently, investigations have examined how the polarity of the solvent influences the degree of  $\pi$ - $\pi$  interactions in violanthrone aggregates.<sup>60</sup> These efforts have revealed that non-polar solvents promote  $\pi$ - $\pi$  interaction, which is in contrast with other aromatic  $\pi$ -stacking systems. However, little work has been done to understand how the structural characteristics of violanthrone aggregates are influenced by solvent effects or how the aggregate structural characteristics influence the movement of vibrational energy through the aggregate. In this work, natural vibrational modes of violanthrone-79 (V-79), shown in figure 4.1, are used to characterize the aggregate structures formed in chloroform (CHCl<sub>3</sub>) and tetrahydrofuran (THF) solvents.

Vibrational spectroscopy experiments, including Fourier-transform infrared (FTIR) spectroscopy and two-dimensional infrared (2DIR) spectroscopy, are powerful techniques to observe the struc-





**Figure 4.1:** Chemical structure of violanthrone-79 (V-79). The vibrational spectrum of V-79 is dominated by the carbonyl stretching modes as well as the in-plane ring modes.

tural and dynamic characteristics of a molecular system. The vibrational modes of a molecular aggregate are sensitive to changes in the local solvent environment as well as the aggregate structure itself. 2DIR spectroscopy provides structural information on molecular aggregates through the examination of cross-peaks observed in the spectra. Additionally, 2DIR experiments can be used to probe dynamic properties of a system by monitoring how the peak shapes and intensities change as a function of delay times between pulses in the experiments. 2DIR spectroscopy experiments have been used to better understand the structure of molecular systems such as DNA,<sup>50</sup> proteins,<sup>52</sup> PAHs,<sup>5</sup> and anti-HIV agents.<sup>61</sup> Chemical dynamics investigated with 2DIR spectroscopy have contributed to our understanding of solvent dynamics and structural dynamics of water,<sup>62</sup> dipeptides, and membranes.<sup>63</sup> In this work, we take advantage of these strengths of 2DIR experiments to study molecular aggregates of V-79 in  $\text{HCCl}_3$  and THF solvents. Specifically, we consider the manner in which the aggregate structures influence the vibrational energy delocalization within the V-79 aggregates. In addition, the experimental results are compared to quantum mechanical (QM) vibrational frequency calculations from molecular structures generated using molecular dynamics (MD) simulations. Solvent-dependent aggregate structures of V-79 are observed and changes in the spectra and the vibrational energy transfer between V-79 molecules within the aggregates are discussed.

## 4.3 Experimental and Computational Methods

### 4.3.1 Sample Preparation

V-79 was purchased from Ark Pharma and used without further purification.  $\text{HCCl}_3$  and THF were purchased from Fisher scientific. Samples of V-79 were prepared at 10 mM concentration in each solvent and sonicated to insure full solvation prior to aggregation. Aliquots of stock solutions were placed between two calcium fluoride ( $\text{CaF}_2$ ) windows with a 250  $\mu\text{m}$  spacer. FTIR spectra were collected with 1  $\text{cm}^{-1}$  spectral resolution as an average of 64 spectra using a Bruker Vertex 70 spectrometer.

### 4.3.2 2DIR Spectroscopy

The 2DIR spectrometer used in this study has been described in detail elsewhere.<sup>40,41</sup> Briefly, ultrafast mid-IR pulses were generated using a Ti:Sapphire regenerative amplifier to pump an optical parametric amplifier (OPA). The regenerative amplifier produces <50 fs laser pulses centered at 790 nm with an average pulse energy of 2.7 mJ at a repetition rate of 1 kHz. The OPA generates mid-IR light centered at 5800 nm with 9  $\mu\text{J}$  of energy per pulse. A 90:10 beam splitter is used to direct 90% of the mid-IR light to a home-built pulse shaper used to generate the pulse pairs with the specific time-delays required to pump the sample and generate third-order signal. The remaining 10% of the mid-IR light from the OPA is used as a probe-pulse. An off-axis parabolic mirror is used to focus the pump and probe beams onto the sample in the pump-probe beam geometry. A retro-reflector set on a computer-controlled stage is used to control the time-delay of the probe pulse relative to the pump-pulses. The computer controlled time-delay is also used to control the timing between the second and third laser pulses when collecting 2DIR spectra as a function of coherence time,  $T_w$ . A second off-axis parabolic mirror collimates the third-order signal field emitted from the sample and directs it into a Horiba Triax 190 spectrometer. The spectrometer is used to frequency resolve the signal across a 64-element mercury cadmium telluride array detector. The resulting spectral resolution is  $\sim 5 \text{ cm}^{-1}$ .

The acousto-optic modulator mid-IR pulse shaper is used to control the amplitude and phase of the individual frequencies in the mid-IR optical pulse.<sup>64-66</sup> This allows the relative phase- and time-delays between the pulses generated in the pulse-pair to be altered with each acoustic wave. The pump-pulse delay was scanned from 0 ps to 2.5 ps with 0.025 ps steps and a 1400  $\text{cm}^{-1}$

rotating frame was applied to allow for faster data collection.<sup>66</sup>  $T_w$ -dependent 2DIR spectra were collected from 0 ps to 7.5 ps. For the first 4 ps the  $T_w$  step size was 0.1 ps, for 4–7.5 ps the steps were changed to 0.5 ps steps. Additional pulse-shaping was used in some data sets to remove contributions from quantum beating and coherence transfer. The pump-pulses were clipped at  $1615\text{ cm}^{-1}$  to only pump the vibrational modes above  $1615\text{ cm}^{-1}$ . The pump-pulses were truncated in the frequency domain by blocking a portion of the beam at the Fourier plane of the pulse shaper. This approach allowed for the carbonyl normal mode to be pumped without pumping the normal modes associated with the ring vibrations of V-79.

Polarization dependent 2DIR experiments were conducted on V-79 aggregates in both solvents to determine relative angles between vibrational normal modes. For these experiments, 2DIR spectra were collected in both XXXX and XXYY polarization configurations. The polarizations were controlled using a half-waveplate and wire grid polarizer combination in the pump beam path to rotate the beam  $90^\circ$ . Furthermore, a wire grid polarizer was placed in the probe beam path to insure a single polarization of mid-IR light was reaching the sample.

### 4.3.3 Molecular Dynamics Simulations

The computational methods employed in this work have been described previously.<sup>5</sup> MD simulations were performed using Replica Exchange Umbrella Sampling (REUS) in the NAMD molecular dynamics software package,<sup>42</sup> to sample the potential energy surface of dimerization between two V-79 molecules in  $\text{HCCl}_3$  and THF as a function of center-of-mass distance between the two molecules. V-79 was parameterized using GAFF<sup>15</sup> and charges were found with RESP;<sup>43</sup> for solvents the AMBER  $\text{HCCl}_3$  box was used,<sup>44</sup> and a self-parameterized THF model was used. Harmonic biasing potentials were placed  $0.5\text{ \AA}$  apart along the collective variable axis and were given spring constants of  $20.0\text{ kcal mol}^{-1}\text{ \AA}^{-2}$ . The resultant center-of-mass distances were analyzed using the weighted histogram analysis method (WHAM)<sup>45</sup> to build the potential of mean force (PMF). The PMF was constructed using a home built binning script to check for ergodicity among the biased windows. Each window in the REUS simulations was run for 100 ns in the NVT ensemble.

#### 4.3.4 Quantum Calculations

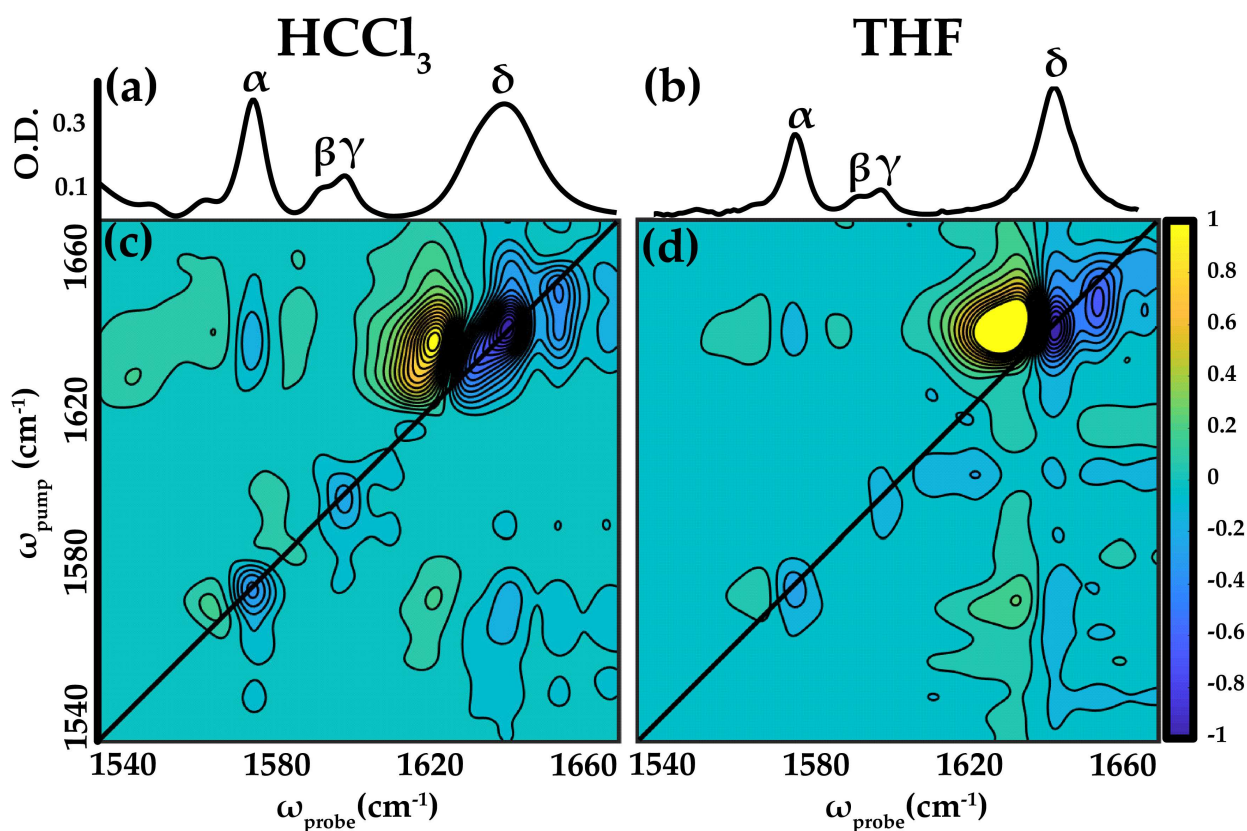
Vibrational frequency calculations were performed using APFD 6-311g\* in Gaussian 16<sup>67</sup> for the V-79 dimer structures that were representative of the two most probable  $\pi$ -stacked dimer configurations. The APFD functional was used for its dispersive properties to stabilize  $\pi$ -stacking interactions between the two monomers. During the geometry optimization four atoms on each molecule were frozen to keep the relative orientation of the molecules stable; the atoms chosen were not active in the pertinent normal modes being studied in this work. As stated in our previous work,<sup>5</sup> this will lead to imaginary frequencies in vibrational modes comprised of these atoms, so we also gave each frozen atom a mass of 1000 AMU to effectively remove its contribution from the vibrational frequency calculation. As a result, each of the V-79 dimer structures were effectively frozen in their initial configurations, able to rotate or translate minimally relative to one another. The protocol described here is essential because the relative orientations of the monomers in the dimer configuration govern the strength and nature of vibrational coupling between the monomer units. Without instituting this position constraining procedure, the dimers change relative orientations and positions significantly during the geometry optimization and differences in relative orientations and thus the resulting vibrational spectra are lost.

#### 4.4 Results and Discussion

The linear FTIR spectra of 10 mM V-79 in  $\text{HCCl}_3$  and THF are shown in figure 4.2a,b respectively. Each spectrum contains four distinct spectroscopic features, denoted  $\alpha$ ,  $\beta$ ,  $\gamma$ , and  $\delta$ . This investigation focuses on the features  $\alpha$  and  $\delta$  which correspond to the in-plane ring stretching motions and carbonyl stretching motions of V-79, respectively. Spectral features  $\beta$  and  $\gamma$  are ignored because of their relatively low intensity.

Comparing  $\delta$  in both solvents reveals clear differences in absorption intensity, width, and center frequency of the peaks. There are several possible scenarios that may give rise to the observed differences in these peaks. The most likely scenarios are: V-79 exists as different aggregate structures in these two different solvent environments, or the local environment surrounding the carbonyl bonds in V-79 is simply more homogeneous in THF as compared to  $\text{HCCl}_3$ , or the carbonyl stretching modes in V-79 are reporting on both contributions—changes in aggregate structure and

changes in the local environment. In THF,  $\delta$  is fit to a Lorentzian function with a full width at half max (FWHM) of  $10.3 \pm 0.25 \text{ cm}^{-1}$  and a center peak position of  $1647 \pm 1 \text{ cm}^{-1}$  (figure 4.6). In contrast, the line shape of  $\delta$  for V-79 aggregates in  $\text{HCCl}_3$  suggests the carbonyls in the aggregate exist in a more inhomogeneous local environment as evidenced by the Gaussian line shape centered at  $1641 \text{ cm}^{-1}$  with a FWHM of  $13.89 \pm 0.09 \text{ cm}^{-1}$  (figure 4.6). MD simulations and QM calculations were used to investigate the effect of aggregate structure on the observed spectral features. Theory and experiment combine to provide a better understanding of the spectral differences in the two solvents.



**Figure 4.2:** FTIR and 2DIR spectra of V-79 in (a,c)  $\text{HCCl}_3$ , and (b,d) THF. Each sample was prepared at a 10 mM concentration.

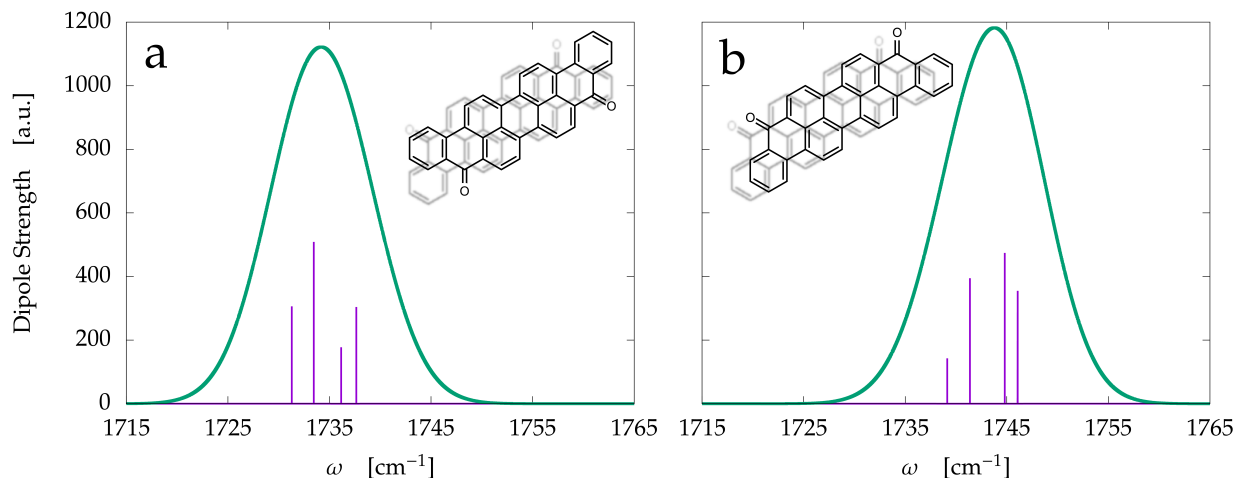
The 2DIR spectra of V-79 in  $\text{HCCl}_3$  and THF are shown in figures 4.2c,d respectively. Within each spectrum there are three spectral features of interest. The diagonal peaks corresponding to  $\alpha$  and  $\delta$  are located at approximately  $\omega_{\text{pump}} = \omega_{\text{probe}} = 1580 \text{ cm}^{-1}$  and  $\omega_{\text{pump}} = \omega_{\text{probe}} = 1640 \text{ cm}^{-1}$  respectively. The differences in peak shape and intensity of the 2DIR feature located at  $\omega_{\text{pump}} =$

$\omega_{\text{probe}} = 1640 \text{ cm}^{-1}$  is related to the differences in line shape discussed above. The third feature in the 2DIR spectra that is a focus of this investigation is the cross-peak between the  $\alpha$  and  $\delta$  modes located at the spectral coordinates  $\omega_{\text{pump}} = 1640 \text{ cm}^{-1}, \omega_{\text{probe}} = 1580 \text{ cm}^{-1}$ . The intensity of this cross-peak relative to the diagonal feature located at  $\omega_{\text{pump}} = \omega_{\text{probe}} = 1640 \text{ cm}^{-1}$  will be used to inform the vibrational dynamics in the V-79 aggregates formed in each solvent.

MD simulations and QM frequency calculations were performed on V-79 dimers to visualize potential aggregate structures and what spectral features each structure exhibited. The PMF of V-79 dimerization in  $\text{HCCl}_3$  and THF were generated from REUS MD simulations as a function of center-of-mass distance between monomers. Each PMF is dominated by one minimum at  $3.6 \text{ \AA}$  separation distance shown in figure 4.7. The observed center-of-mass separation is indicative of a  $\pi$ -stacked dimer. There is no evidence of non-ergodic sampling between the bias windows, thus we are confident in this result. Further orientational analysis was done on the structures found at the PMF minimum corresponding to the dimer state to extract information regarding the molecular geometries present in the PMF minimum of the V-79 aggregates in each solvent. The orientational analysis was composed of six orthogonal helical parameters as in our previous work.<sup>5</sup> All structures in the dimer state in both  $\text{HCCl}_3$  and THF fell under narrow unimodal distributions in all helical parameters except for one bimodal distribution corresponding to the presence of both antiparallel and parallel  $\pi$ -stacking dimer structures. In the parallel geometry the alkyl tails of both molecules are on the same side of the  $\pi$ -stacked cores and the antiparallel geometry has one monomer rotated  $180^\circ$  such that the alkyl tails are on opposite sides of the  $\pi$ -stacked cores, figure 4.3 insets. The MD simulations showed the ratio of antiparallel to parallel stacks are solvent dependent, with a ratio of 11:1 in  $\text{HCCl}_3$  and 2.5:1 in THF, antiparallel to parallel. Based on the difference in ratio of antiparallel to parallel stacks found in MD simulations, it can be hypothesized that the difference in experimentally observed spectra is originating from differing ratios of antiparallel and parallel  $\pi$ -stacks in  $\text{HCCl}_3$  and THF.

QM frequency calculations were performed on the antiparallel and parallel dimer structures to determine the differences in their vibrational spectra. The results of the QM frequency calculations on the antiparallel and parallel dimer configurations isolated from MD simulations are shown in figures 4.3a,b respectively. These spectra are truncated to show only the modes related to the carbonyl region of the dimer; the calculated frequencies have not been shifted, because only

the relative position of the vibrational energies are considered. These results show each dimer configuration contains four different carbonyl vibrational modes. However, the antiparallel configuration's center frequency is lower,  $1735\text{ cm}^{-1}$ , than the parallel configuration,  $1745\text{ cm}^{-1}$ . This shift is consistent with the experimental results, figure 4.2b, which showed  $\text{HCCl}_3$  with a lower frequency carbonyl stretch,  $\delta$ , than the corresponding mode in THF. This supports our findings from MD that  $\text{HCCl}_3$  promotes antiparallel stacking more than THF. Additionally, the carbonyl motions in the antiparallel dimer are localized to a single monomer for each normal mode in the QM frequency calculations whereas the carbonyl motions in the parallel dimer are delocalized across the dimer for each normal mode. This implies that vibrational energy is localized across the aggregate in the antiparallel  $\pi$ -stack and delocalized for the parallel  $\pi$ -stack. 2DIR experiments were used to investigate the extent of energy delocalization in the V-79 aggregates formed in  $\text{HCCl}_3$  and THF.



**Figure 4.3:** Vibrational spectra from QM frequency calculations for (a) the antiparallel and (b) the parallel V-79 dimer configurations. Each spectrum contains four vibrational modes dominated by the carbonyl stretching motions in the dimer depicted by the stick spectrum. The stick spectra are convoluted with Gaussian line widths with  $\text{FWHM} = 10\text{ cm}^{-1}$ . The inset images depict the general arrangements of the respective dimers.

Vibrational energy delocalization can be observed experimentally through examining population transfer with 2DIR. Population transfer within V-79 molecular aggregates was studied by considering the cross relaxation between vibrational modes. The orientation of the molecules within an aggregate will fluctuate thereby producing fluctuations in the coupling between their

vibrational modes. This can be represented with the time-dependent Hamiltonian shown in equation (4.1).

$$H(t) = \begin{pmatrix} \hbar\omega_A + \hbar\delta\omega_A(t) & \beta_{AB} + \delta\beta_{AB}(t) \\ \beta_{AB} + \delta\beta_{AB}(t) & \hbar\omega_B + \hbar\delta\omega_B(t) \end{pmatrix} \quad (4.1)$$

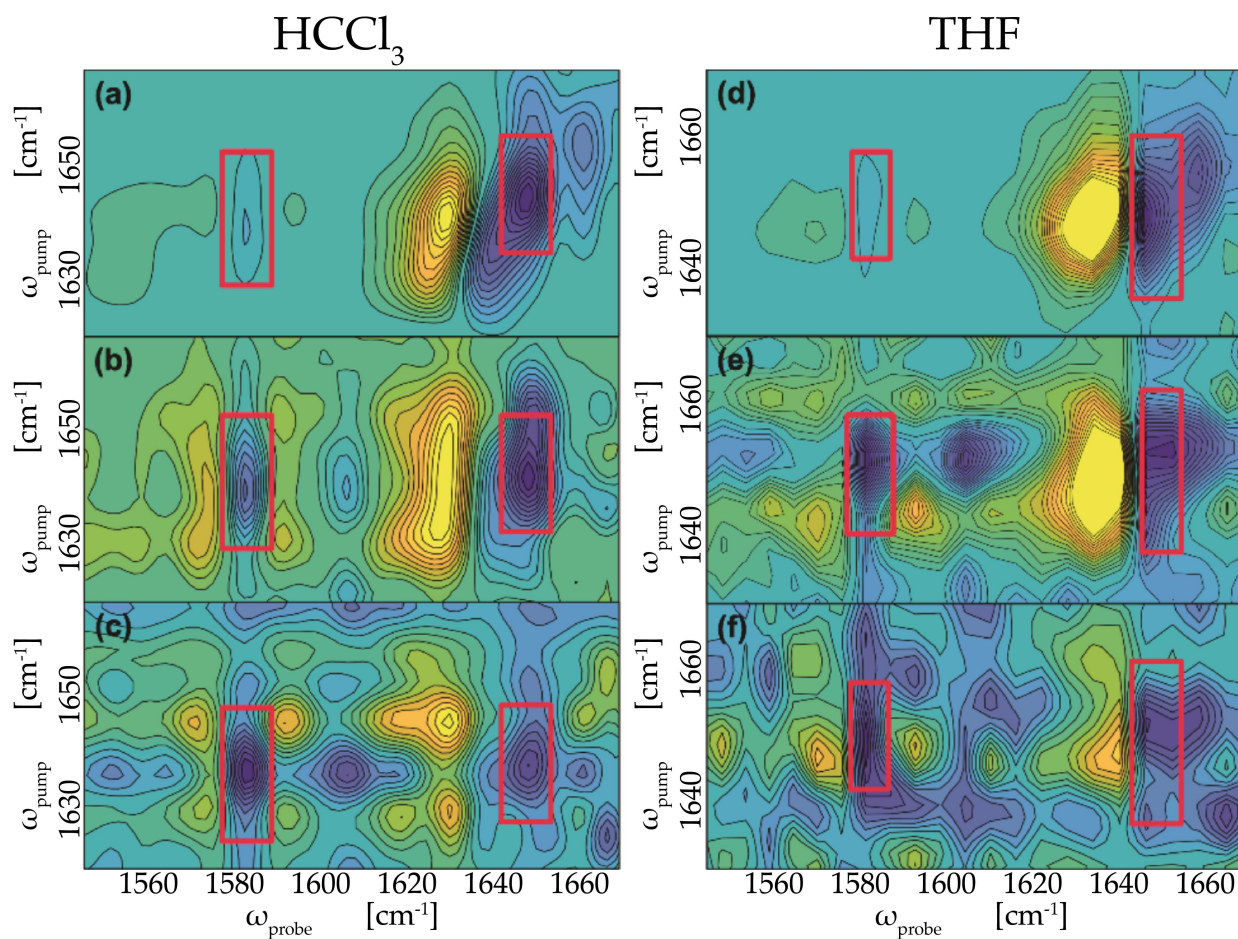
where  $\delta\omega_A$  and  $\delta\omega_B$  are the time-dependent fluctuations in the frequencies of modes  $A$  and  $B$  with mean frequencies  $\omega_A$  and  $\omega_B$  respectively.  $\beta_{AB}$  is the vibrational coupling term between modes  $A$  and  $B$  with time-dependent fluctuations  $\delta\beta_{AB}$ . The fluctuations in the coupling term result in changes in the relative intensity between cross- and diagonal-peaks as a function of probe delay time,  $T_w$ . Quantifying this relative intensity change can provide the time scales of vibrational energy transfer through an aggregate. The polarization-dependent 2DIR measurements indicated the angle between the vibrational modes of interest were not  $90^\circ$  in either solvent. Thus, the vibrational coupling in these aggregates is assumed to be in the weak coupling limit. Having made this distinction, a rate of energy transfer can be derived from equation (4.1) by diagonalizing the Hamiltonian and assuming the weak coupling limit.<sup>68</sup>

$$k = \frac{2(d/\hbar)^2\tau}{1 + \hbar(\Delta\omega/\hbar)^2\tau^2} \quad (4.2)$$

Time-dependent clipped pump 2DIR experiments were used to compare vibrational energy transfer within the two aggregate systems. The pump spectra were clipped such that the ring mode  $\alpha$  was not pumped, removing contributions from coherence transfer in the cross-peak intensity. Representative clipped pump spectra for  $T_w$  equal to 0, 3, and 5 ps are shown in figure 4.4 for V-79 aggregates formed in  $\text{HCCl}_3$  (figure 4.4a-c) and THF (figure 4.4d-f). In figure 4.4 the red boxes indicate the integrated areas used to determine the change in peak intensity plotted in figure 4.5. The ratio of the integrated intensity between the cross-peak located at the spectral coordinates,  $\omega_{\text{pump}} = 1640 \text{ cm}^{-1}$ ,  $\omega_{\text{probe}} = 1580 \text{ cm}^{-1}$ , and the diagonal peak at  $\omega_{\text{pump}} = \omega_{\text{probe}} = 1640 \text{ cm}^{-1}$  was calculated at each time step. The ratio of integrated intensities is used to track the relative change in intensity, thereby taking into account the loss in intensity from the vibrational life time.

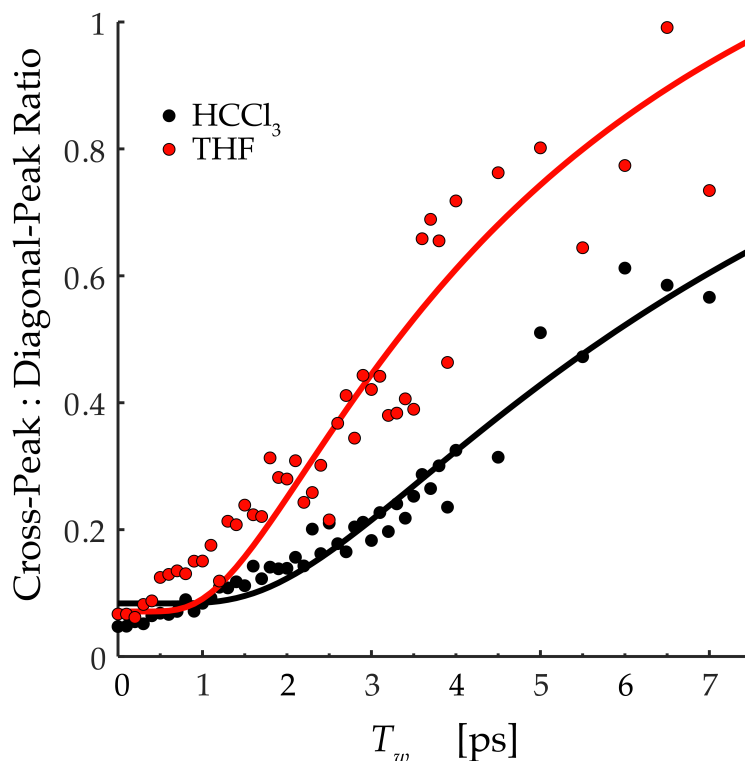
Plots of the ratios of integrated intensities as a function of  $T_w$  are shown in figure 4.5. Exponential fits of the data in figure 4.5 give cross relaxational energy transfer rates of  $7.2 \pm 1.4 \text{ ps}^{-1}$  in  $\text{HCCl}_3$





**Figure 4.4:** Clipped pump 2DIR spectra of the carbonyl region and cross-peak. The red boxes indicate the regions that were integrated to determine the peak ratio between the cross-peak and the diagonal-peak. Each row corresponds to a different  $T_w$  time **(a,d)** 0 ps, **(b,e)** 3 ps, and **(c,f)** 5 ps.

and  $4.4 \pm 1.2 \text{ ps}^{-1}$  in THF. The slower rate observed in  $\text{HCCl}_3$  systems suggests vibrational energy is more localized in these aggregates in comparison to those in THF. Therefore, based on the delocalization of the atomic motions in the QM frequency calculations of the parallel dimer, we would expect to find more parallel aggregate structures in THF than in  $\text{HCCl}_3$  which is supported by our findings from MD.



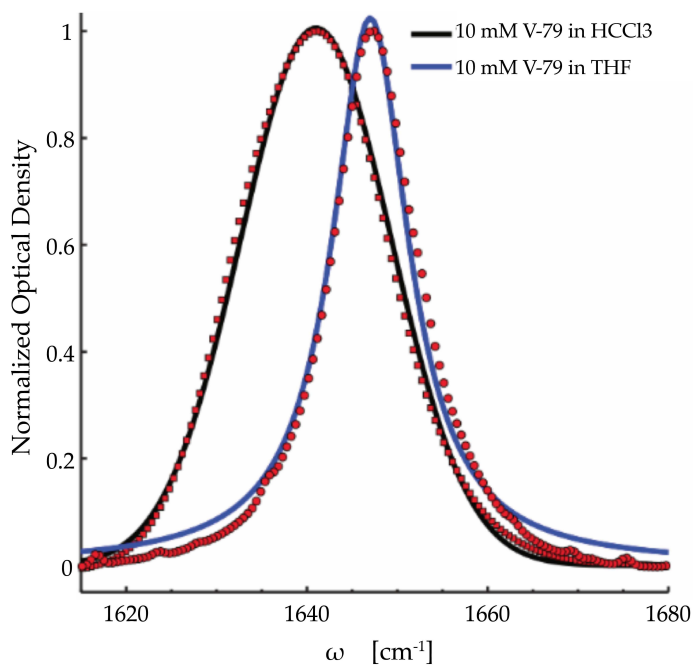
**Figure 4.5:** Ratio of the integrated intensity between the diagonal peak and cross-peak in the 2DIR spectra of V-79 in THF (red) and  $\text{HCCl}_3$  (black). The points are the experimental values and the lines are generated by fitting the data to extract the time for cross relaxational energy transfer in the aggregates of V-79.

## 4.5 Conclusion

The collective results from the experiments, MD simulations, and QM calculations in this work have revealed the propensity of each solvent,  $\text{HCCl}_3$  and THF, to support the formation of different V-79 self-assembled structures.  $\text{HCCl}_3$  is found to support the formation of antiparallel  $\pi$ -stacking V-79 aggregates, while THF is found to facilitate the formation of parallel  $\pi$ -stacking V-79 aggregates. Furthermore, the QM calculations demonstrated that vibrational energy was localized in antiparallel aggregates; in contrast, vibrational energy of the carbonyl stretching modes are delocalized to the V-79 monomers in the parallel dimers. Finally, time-dependent 2DIR measurements showed notably slower vibrational energy delocalization in V-79 aggregates formed in  $\text{HCCl}_3$  compared to THF. The hypothesis that  $\text{HCCl}_3$  supports the formation of V-79 aggregates that have significantly more antiparallel  $\pi$ -stacks, whereas THF supports the formation of V-79 aggregates that are relatively split antiparallel and parallel  $\pi$ -stacking interactions is confirmed.

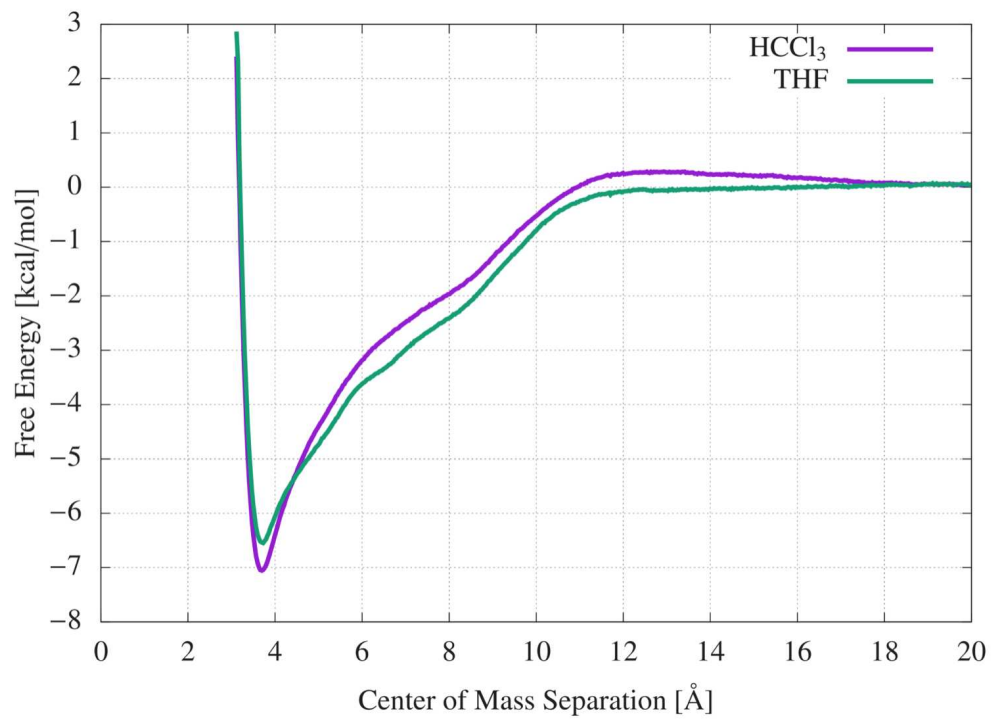
Thus, solvent choice can be a parameter used to tailor aggregate structures to specific applications in photochemical technologies.

## 4.6 Supporting Information



**Figure 4.6:** FTIR spectra of the carbonyl spectral region of V-79 aggregates in  $\text{HCCl}_3$ , black, and in THF, blue. The spectral feature of the aggregate formed in  $\text{HCCl}_3$  is best fit using a Gaussian line shape. The spectral feature of the aggregate formed in THF is best fit to a Lorentzian.

Figure 4.7 shows the PMFs of V-79 in  $\text{HCCl}_3$  and THF as a function of center of mass distance between the V-79 molecules. Both PMFs have their global minimum at 3.6 Å indicating a  $\pi$ -stacked structure. All orientational analysis was done on dimer structures within this minimum.



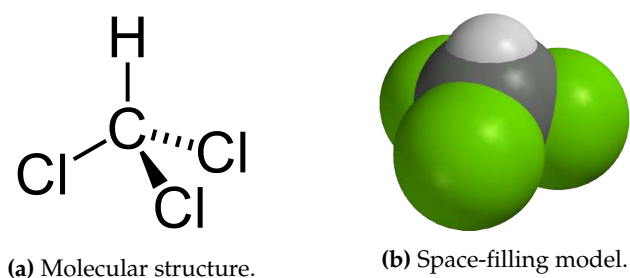
**Figure 4.7:** PMFs of V-79 in HCCl<sub>3</sub> and THF as a function of center of mass separation distance between the two V-79 molecules.

## Chapter 5

### Development of IS-SPA for Chloroform

#### 5.1 Non-Spherical Solvent

Development of IS-SPA for chloroform is the first time IS-SPA is being used for a solvent that is more than a single spherically symmetric LJ potential. IS-SPA was originally developed for the Transferable Intermolecular Potential with 3 Points (TIP3P) model of water.<sup>12</sup> TIP3P has a static geometry with point charges corresponding to each atom and a single LJ potential for the whole molecule centered on the oxygen atom. Chloroform, figure 5.1, has five atoms all of which have LJ potentials and charges in the model used in this work.<sup>14</sup> The bonds and angles of chloroform are flexible allowing the molecule to deform from its equilibrium structure. Because of the smaller size of the hydrogen, the chlorines lie in nearly the same plane as the carbon atom giving the molecule an almost disk-like shape with  $C_{3v}$  symmetry about the CH axis. The hydrogen is positively charged at  $0.266 q_e$ , the carbon is negatively charged at  $-0.385 q_e$ , and the chlorines are negligibly charged at  $0.040 q_e$  giving this model of chloroform a dipole moment of  $0.2915 q_e \text{ \AA}$  where  $q_e$  is the elemental charge. The LJ parameters for each atom are detailed in table 5.1



**Figure 5.1:** The structure of chloroform the model solvent used to expand the IS-SPA theory to non-spherical solvent molecules.

There are three notable qualities that chloroform possesses that were either absent from TIP3P or not addressed in the original development of IS-SPA for TIP3P: a flexible internal structure of the solvent, a non-spherical LJ potential brought on by multiple atoms, and the presence of a dipole in

<sup>12</sup>Jorgensen, W. L. et al. *The Journal of Chemical Physics* **1983**, 79, 926–935.

<sup>14</sup>Cieplak, P.; Caldwell, J.; Kollman, P. *Journal of Computational Chemistry* **2001**, 22, 1048–1057.

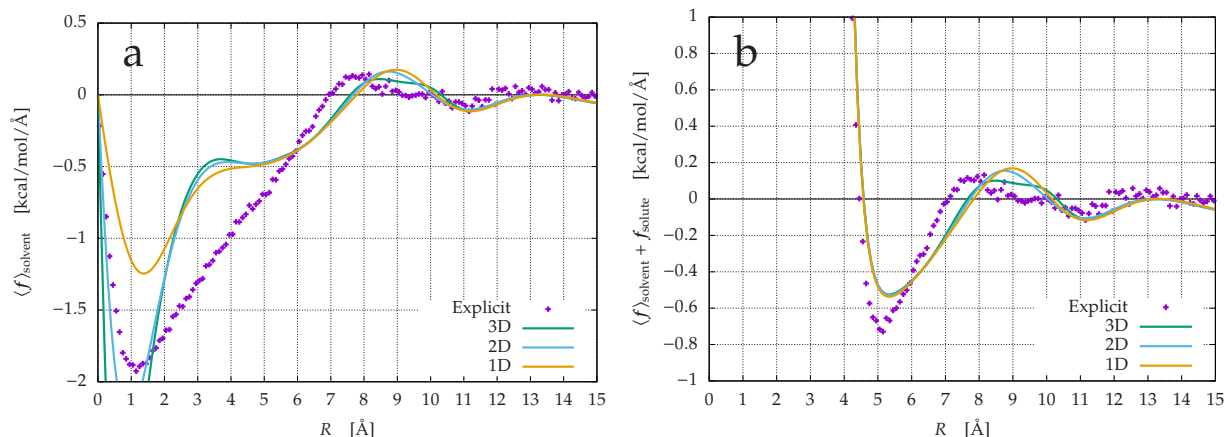
the solvent and its effect on charged solutes. The deformation of the solvent would present itself at close distances where the solvent molecules are packed up against the solute and deformed from equilibrium structure. However, since the distribution and force data we are using is measured directly from simulation this quality should be implicitly captured in our histograms. The use of analytic extrapolation for these functions at small values of  $r$  would be one potential place where this could resurface in a problematic way because equilibrium structure chloroform is used for the analytic extrapolation. However, for all cases we have encountered, the analytic extrapolation has smoothly transitioned the histograms to their respective end-behaviors without noticeable kinks in the transition from measured to ideal data.

**Table 5.1:** AMBER chloroform LJ parameters.

atom	<b>A</b> [kcal/mol/Å <sup>12</sup> ]	<b>B</b> [kcal/mol/Å <sup>6</sup> ]	$\epsilon$ [kcal/mol]	$r_{\min}$ [Å]
H	$5.031 \times 10^2$	$5.621 \times 10^0$	0.016	2.374
C	$1.043 \times 10^6$	$6.756 \times 10^2$	0.109	3.816
Cl	$5.452 \times 10^6$	$2.662 \times 10^3$	0.325	4.000

Since chloroform is non-spherical it would be reasonable to assume that modeling its non-spherical shape would be important. In IS-SPA we would accomplish this by considering higher dimensionality representations of the solvent as outlined in sections 2.2.2.2 to 2.2.2.4 on pages 11 to 15. However, as shown in figures 5.2 and 5.3, surprisingly the higher dimensional representations have minimal increase in the accuracy of the mean force of the model system and this minimal increase in accuracy of the 3-dimensional or ellipsoidal symmetry system comes at a great price in terms of efficiency. To get a sense of how much more expensive the 3-dimensional model is than the 1-dimensional, to generate the mean force curves for figures 5.2 and 5.3 the 3-dimensional model takes 38 hours running in parallel on 8 CPUs and the 1-dimensional model takes 30 seconds on only 1 CPU. The small improvement to the barrier accuracy of the 3-dimensional model is not worth this extreme difference in efficiency if the added accuracy is not required to capture the correct physics.

The model system simulations were performed without the direct interaction between the two solutes for debugging purposes and because adding in the direct solute-solute force afterwards



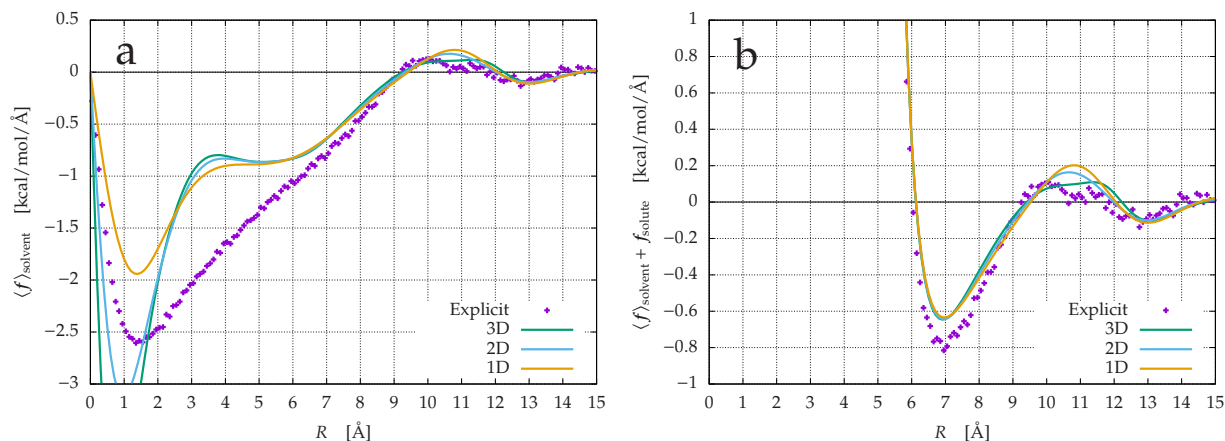
**Figure 5.2:** The 1, 2, and 3-dimensional distribution and force histogram mean forces are overlaid on the explicit MD mean force. **(a)** Mean solvent force  $\langle f \rangle_{\text{solvent}}$  and **(b)** total force  $\langle f \rangle_{\text{solvent}} + f_{\text{solute}}$  as a function of solute separation distance  $R$  in the  $r_{\min} = 5.0 \text{ \AA}$  model system.

is trivial. As the solutes start to overlap in figure 5.2a at distances below  $5 \text{ \AA}$  the IS-SPA models begin to deviate significantly from the explicit. While this may appear to be a problem at first glance all of the deviations at these close distances would, in the case of real solute atoms, be occurring at large repulsive forces when the direct interaction term is included as in figure 5.2b. So the important range for the models to be accurate over is all distances larger than  $\sim 4 \text{ \AA}$  in the case of  $r_{\min} = 5 \text{ \AA}$  solutes. This is evidenced by the steep incline seen in figure 5.2b where the force increases rapidly at distances below  $\sim 5 \text{ \AA}$ . Any disagreement between IS-SPA and explicit at distances below this are occurring at large repulsive forces that have minimal probability of being sampled in simulation.

For the  $r_{\min} = 5 \text{ \AA}$  model system all of the IS-SPA mean forces overestimate the position of the repulsive barrier by about  $1 \text{ \AA}$ . While the 3-dimensional histogram does reproduce the repulsive force barrier the best of all the models—it mirrors the explicit with a slightly flatter barrier than the other two—the total difference between all three models is not significant. All three dimensionalities have their repulsive barriers at the same overestimated distance and they all tend towards the same values at large  $R$  with the overall shape of the oscillations from AESMD being reproduced.

One hypothesis for this overestimation of the distance at which the barrier occurs is that the solutes are about the same size as the chloroform molecule and that perhaps there is some minimum relative size the solute has to be for our model to work correctly. To test this hypothesis we

conducted the same analysis on a model system with  $r_{\min} = 7 \text{ \AA}$ . The commensurate mean force figures for this system are shown in figure 5.3.



**Figure 5.3:** The 1, 2, and 3-dimensional distribution and force histogram mean forces are overlaid on the explicit MD mean force. **(a)** Mean solvent force  $\langle f \rangle_{\text{solvent}}$  and **(b)** total force  $(\langle f \rangle_{\text{solvent}} + f_{\text{solute}})$  as a function of solute separation distance  $R$  in the  $r_{\min} = 7.0 \text{ \AA}$  model system.

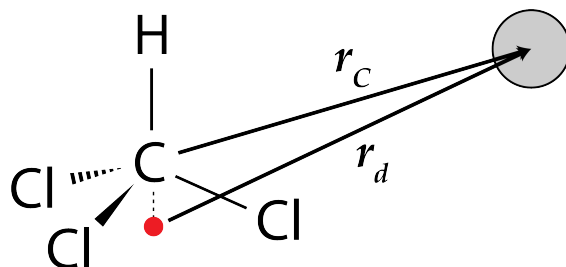
The barrier position problem is improved in the  $r_{\min} = 7 \text{ \AA}$  solute system suggesting that there may be a minimum solute size limitation to the model, however further simulations as a function of  $r_{\min}$  distance would have to be performed to demonstrate this fully.

## 5.2 1-Dimensional Histogram Considerations

Since all three symmetry representations of chloroform are very similar in terms of the mean forces they predict we have chosen to use the 1-dimensional or spherical representation because it is the simplest to implement and the most efficient. The 1-dimensional representation effectively treats chloroform as a sphere by using distribution and force histograms,  $g(r)$  and  $f(r)$  respectively, that only depend on separation distance  $r$ . However, since chloroform is not a sphere there is the question of where to measure the center of the solvent from. A first approximation is the carbon atom as this would be the center of a homonuclear tetrahedral molecule, but since the hydrogen has a significantly smaller Van der Waals radius than the chlorines, the carbon actually lies closer to the hydrogen side of the excluded-volume of the chloroform molecule. To explore the significance of the center position of chloroform we have run 1-dimensional simulations with distribution and

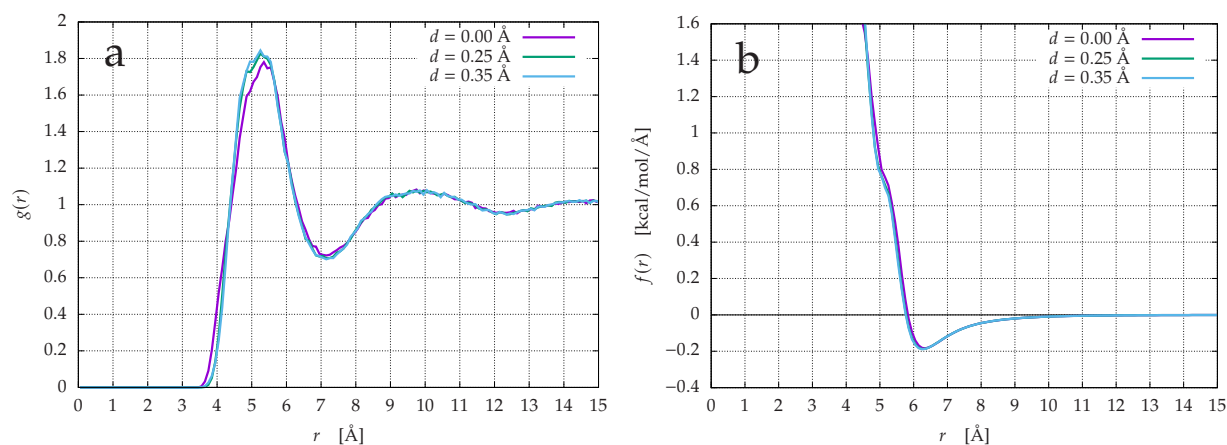


force data collected using a center position that is offset from the carbon towards the chlorines as depicted in figure 5.4.

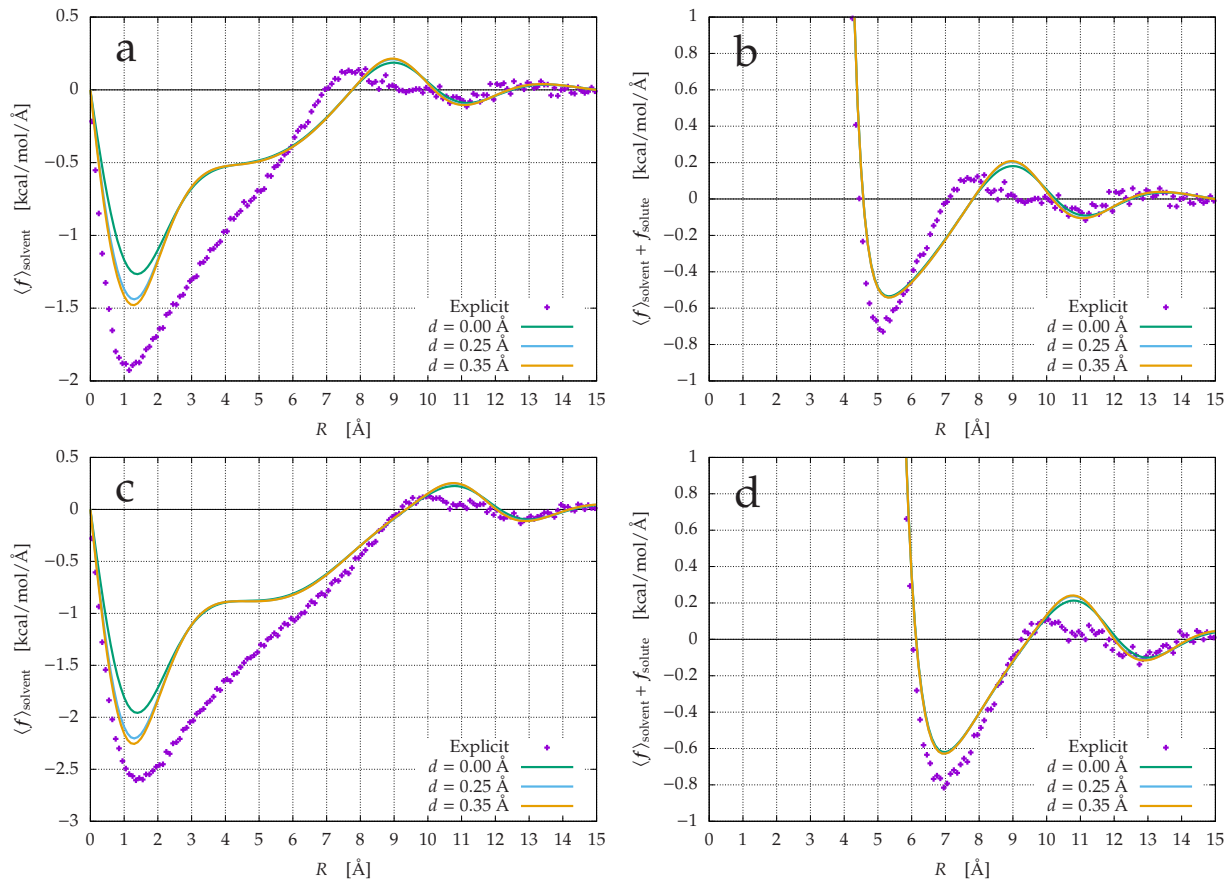


**Figure 5.4:** The solvent–solute separation vector  $r$  shown from carbon center,  $r_C$ , and excluded offset center,  $r_d$ , where the red point represents the new center of excluded volume of the chloroform some distance  $d$  offset from the carbon in the direction of the chlorines.

The difference in the distribution and force as a function of offset distance  $d$  is minimal as seen in figure 5.5. There are however some minor differences in the peak height and shape of the distribution function in figure 5.5a. To assess the impact of these minor differences the mean force as a function of offset should be examined. Figure 5.6 shows the mean force as a function of solute–solute separation distance for the 1-dimensional histogram representation for both  $r_{\min} = 5$  Å and  $r_{\min} = 7$  Å solute sizes. The three offset distance curves are almost indistinguishable at relevant distances, ( $r > 4$  Å and  $r > 6$  Å respectively), which confirms that the offset distance is not an important variable for the LJ interactions in the model system.



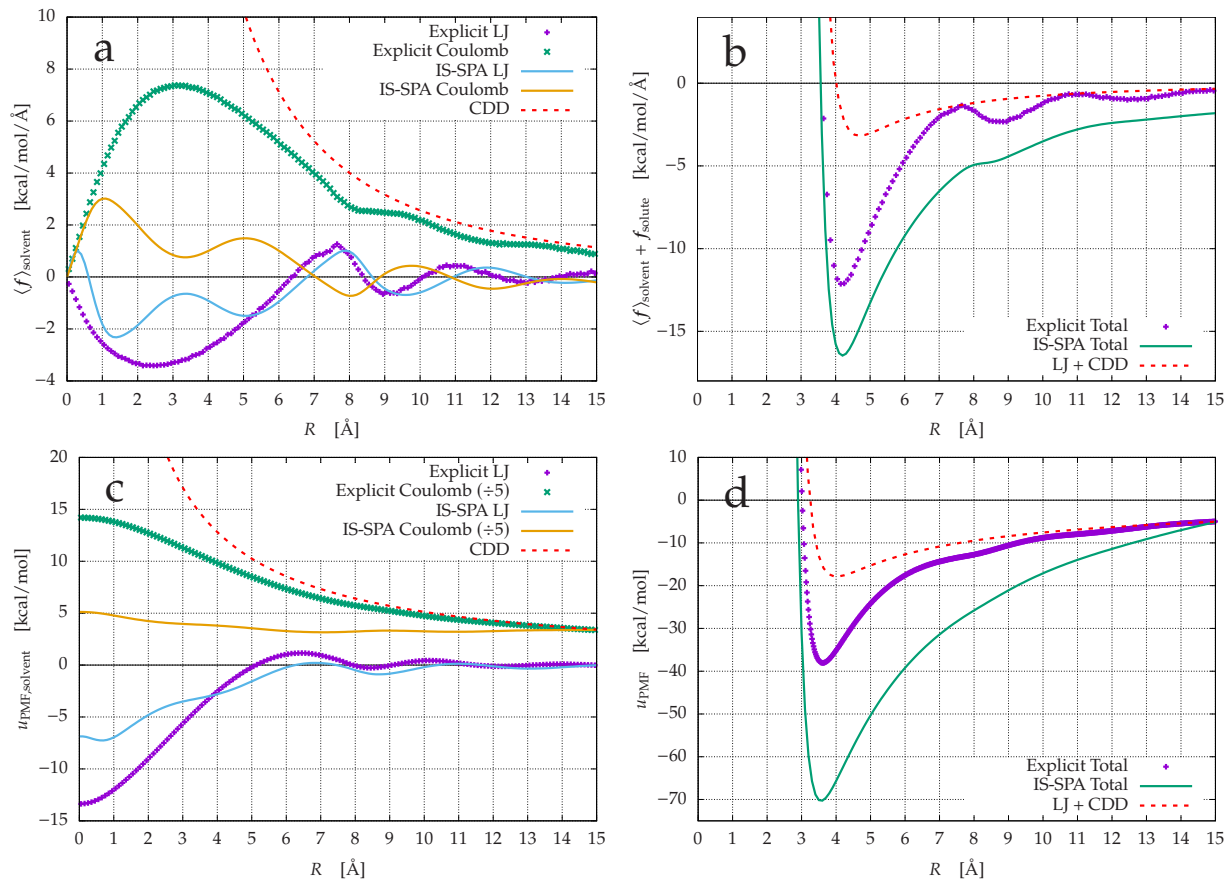
**Figure 5.5:** (a) Distribution  $g(r)$  and (b) force  $f(r)$  functions for chloroform around the  $r_{\min} = 5$  Å solute particle with three different center offset distances,  $d$ .



**Figure 5.6:** 1-dimensional mean force plots for a range of solvent center offsets,  $d$ . (a) Mean solvent force  $\langle f \rangle_{\text{solvent}}$  and (b) total force ( $\langle f \rangle_{\text{solvent}} + f_{\text{solute}}$ ) as a function of solute separation distance  $R$  in the  $r_{\text{min}} = 5.0 \text{ \AA}$  model system. (c) Mean solvent force  $\langle f \rangle_{\text{solvent}}$  and (d) total force ( $\langle f \rangle_{\text{solvent}} + f_{\text{solute}}$ ) as a function of solute separation distance  $R$  in the  $r_{\text{min}} = 7.0 \text{ \AA}$  model system.

To assess whether the accuracy of the 1-dimensional model is sufficient, we must compare the LJ and Coulombic parts and test the model on a molecular system via IS-SPA numerical integration. For these calculations both LJ and Coulombic forces were modeled as 1-dimensional, i.e. spherically-symmetric. In the context of electrostatics, this means that the Coulombic forces were treated as monopolar since they emanate from a point and diverge radially.

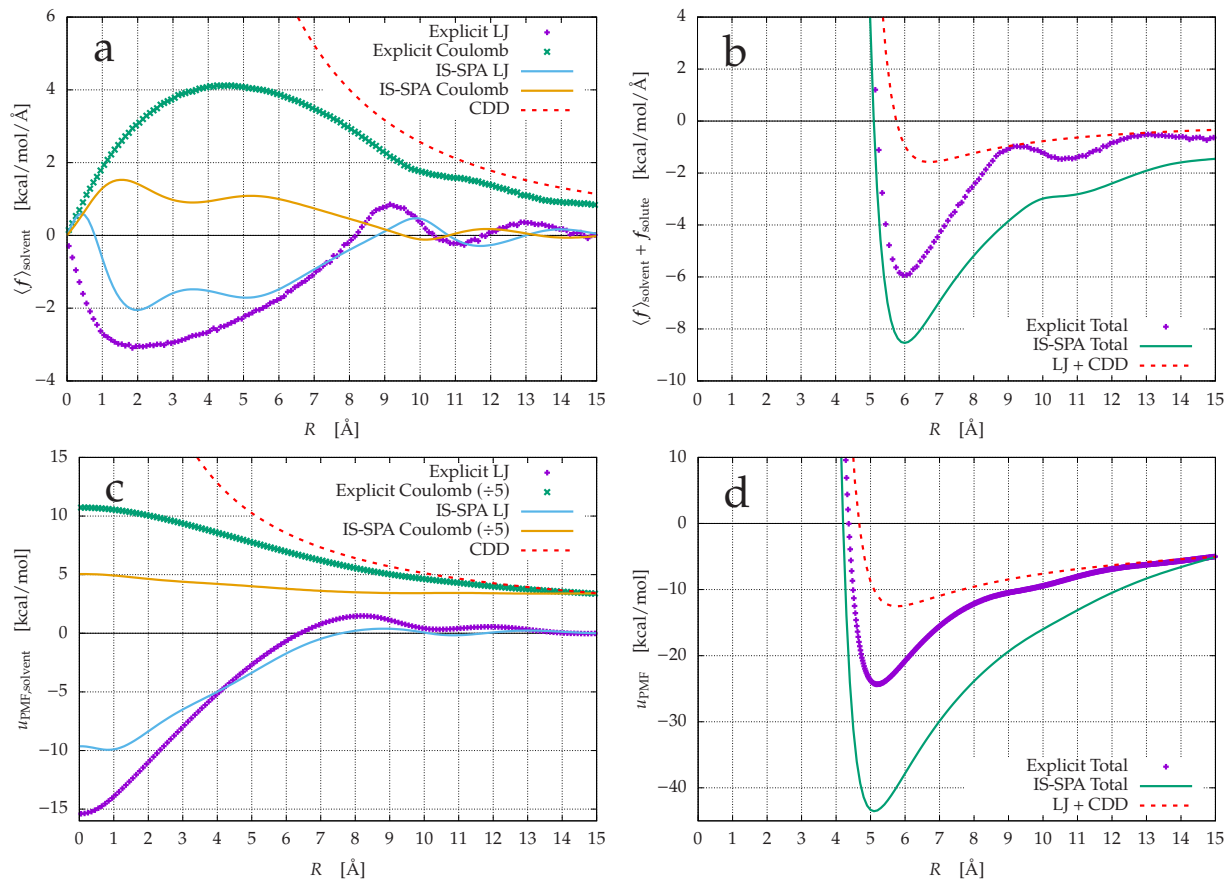
The comparison of each force and PMF for the model system with charge  $\pm 1.0 q_e$  in figures 5.7 and 5.8 for the  $r_{\text{min}} = 5$  and  $7 \text{ \AA}$  solutes respectively. The separated forces in figures 5.7a and 5.8a show that the LJ components of IS-SPA capture the general trends and features of the explicit LJ force. Treating the solvent as radially symmetric electrostatically does not reproduce the repulsive force that is present in the explicit solvent. Since the IS-SPA radial Coulombic force hovers around



**Figure 5.7:** Model system mean force and PMF results of IS-SPA with radial Coulombic forces as a function of solute-solute distance  $R$  compared to AESMD for  $r_{\text{min}} = 5 \text{ \AA}$  and  $q = \pm 1.0 q_e$  solute particles. **(a)** Mean force for the LJ and Coulombic portions separately. **(b)** Total force with direct solute-solute term. **(c)** PMFs of LJ and Coulombic portions separately. **(d)** Total PMF with direct solute-solute term. Each panel also includes a dashed red curve that is the Coulombic force or PMF assuming the solvent is a constant density dielectric which is explained in section 5.3.

zero as a function of  $R$ , the resultant total force and PMF wells are significantly over exaggerated. These differences show that representing chloroform as radially symmetric electrostatically is not sufficient but they do not identify why it is insufficient. This is more clearly seen by analyzing the difference vectors on each atom of our model molecular system LO.

Using our molecular model system LO, we compared the average LJ and Coulombic forces on each solute atom between explicit and IS-SPA, the results of which are shown in figure 5.9. All difference vectors have been scaled up by a factor of 2 so as to be more easily visible for viewing purposes. Examination of the difference in forces shows that the LJ forces are under-predicted in general by IS-SPA with the most solvent exposed external atoms having difference vectors pointing inward from the solvent. This fits with what is expected of LJ forces which are dominated by

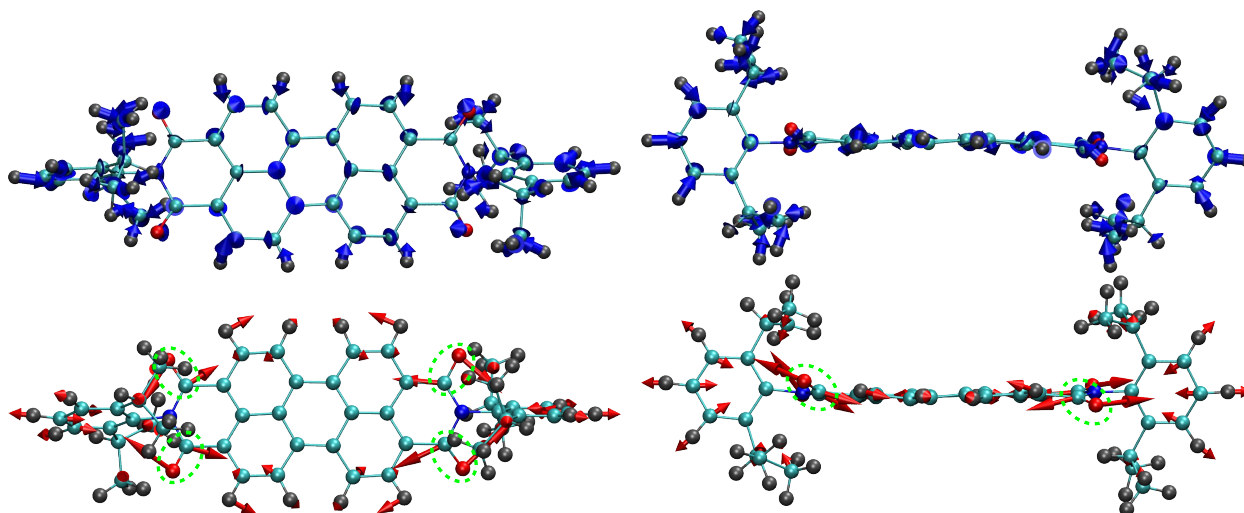


**Figure 5.8:** Model system mean force and PMF results of IS-SPA with radial Coulombic forces as a function of solute-solute distance  $R$  compared to AESMD for  $r_{\min} = 7 \text{ \AA}$  and  $q = \pm 1.0 q_e$  solute particles. **(a)** Mean force for the LJ and Coulombic portions separately. **(b)** Total force with direct solute-solute term. **(c)** PMFs of LJ and Coulombic portions separately. **(d)** Total PMF with direct solute-solute term. Each panel also includes a dashed red curve that is the Coulombic force or PMF assuming the solvent is a constant density dielectric which is explained in section 5.3.

their strongly repulsive behavior from external pressure on the system,<sup>69</sup> which IS-SPA seems to be systematically underpredicting albeit minimally. The Coulombic forces however, are not predicted with the same degree of accuracy, the forces on the most charged atoms deviate in a manner which implies that the Coulombic force requires a dipole component.

The necessity of a dipole component to the Coulombic force is most clearly evidenced by the forces on the negatively charged oxygen and positively charged carbonyl carbon of LO, (circled in green in figure 5.9). The difference vectors of these two atoms point in opposite directions transverse to the majority of the solvent exposed volume around them. The directions of these difference forces are indicative of a dipole force because the electric field of a dipole emanates

<sup>69</sup>Weeks, J. D.; Chandler, D.; Andersen, H. C. *The Journal of Chemical Physics* **1971**, *54*, 5237–5247.



**Figure 5.9:**  $\langle \mathbf{f} \rangle_{\text{solvent}}^{(\text{explicit})} - \langle \mathbf{f} \rangle_{\text{solvent}}^{(\text{IS-SPA})}$  difference vectors for LJ forces (**top**), and Coulombic forces (**bottom**). The green dashed circles are around the carbonyl carbon and oxygen, the two most highly charged atoms in the molecule, which have difference vectors characteristic of an electrostatic dipole force in the Coulombic difference vectors (**bottom**).

from the positive pole to the negative pole in a toroidal fashion. Thus a solvent molecule near the carbonyl group with its dipole oriented perpendicular to the carbonyl bond would produce a field that would pass through both solute atoms perpendicular to the carbonyl bond between them. This field would therefore produce forces on each atom perpendicular to the carbonyl bond and in opposite directions due to the opposite charges on the atoms which is exactly what is seen in figure 5.9. While a 1-dimensional treatment of the LJ forces may be sufficient, it seems as though a more complex treatment of the Coulombic forces is required.

## 5.3 Dipole Coulombic Forces in IS-SPA

### 5.3.1 Polarization Theory

The dipole component of a field is the second term of the multipolar expansion after the monopole term. The electric field of a dipole is described by,

$$E_{\text{dip}}(\mathbf{r}) = \frac{1}{4\pi\epsilon_0} \frac{1}{r^3} (3(\mathbf{p} \cdot \hat{\mathbf{r}})\hat{\mathbf{r}} - \mathbf{p}) \quad (5.1)$$

where  $\hat{\mathbf{r}}$  is the unit vector of the separation vector between the dipole and the point in space,  $\mathbf{r}$ , and  $\mathbf{p}$  is the dipole vector in the physics definition, i.e.  $\mathbf{p}$  points from negative pole to positive pole. So, if we know the direction and magnitude of the solvent dipole at a given position we can calculate

the force on the solute charge via equation 5.1 and the Lorentz force,  $F = qE$ , where  $q$  is the charge of the solute atom. Thus we must find a way to predict what the average solvent polarization vector,  $\mathbf{p}$ , is at a given location,  $\mathbf{r}$ , given solute positions  $\mathbf{R}^N$ .

We begin by defining the energy of a dipole in an external field that will form the Boltzmann factor for our probability distribution. In the presence of an electric field dipoles preferentially orient themselves parallel with the field to minimize their potential energy  $u_{\text{dip}}$ ,

$$u_{\text{dip}} = -\mathbf{E} \cdot \mathbf{p} \quad (5.2)$$

where  $\mathbf{E}$  is the external electric field and  $\mathbf{p}$  is the dipole moment. Using the energy of a dipole in a field we can calculate the probability of a dipole being in a given alignment relative to the external field  $\mathbf{E}$ ,

$$\begin{aligned} \mathcal{P}(\cos \theta) &\propto e^{-u_{\text{dip}}(\cos \theta)/T} \, d\cos \theta \\ &\propto e^{+Ep \cos \theta/T} \, d\cos \theta \end{aligned} \quad (5.3)$$

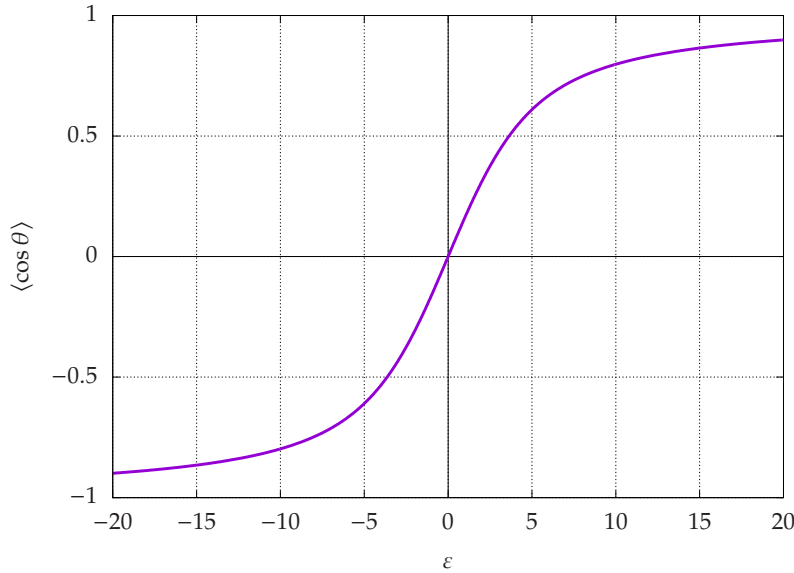
where  $E$  is the magnitude of the electric field,  $p$  is the magnitude of the dipole moment, and  $\theta$  is the angle between  $\mathbf{E}$  and  $\mathbf{p}$ , and  $T$  is the temperature in units of energy. We can determine the average alignment of a dipole with static magnitude,  $p\langle\hat{\mathbf{p}}\rangle = p\langle\cos \theta\rangle$ , in a field by solving for  $\langle\hat{\mathbf{p}}\rangle = \langle\cos \theta\rangle$ ,

$$\langle\cos \theta\rangle = \frac{\int_{-1}^{-1} d\cos \theta \cos \theta \mathcal{P}(\cos \theta)}{\int_{-1}^{-1} d\cos \theta \mathcal{P}(\cos \theta)} \quad (5.4)$$

$$= \coth(\varepsilon) - \frac{1}{\varepsilon} \quad (5.5)$$

where the final result is the Langevin function of the reduced-field  $\varepsilon = Ep/T$  which is plotted in figure 5.10. At small field strengths the dipoles respond linearly, aligning more and more as the field builds in strength. However, as the field continues to grow in magnitude the polarization

levels off to a value of one. This is because in our treatment above we assumed a static dipole moment, i.e. the dipole cannot stretch or compress, so the maximum polarization is that of complete alignment i.e.,  $\langle \cos \theta \rangle = \pm 1$ , depending on the sign of the field.



**Figure 5.10:** The mean polarization,  $\langle \cos \theta \rangle$ , of a static dipole follows the Langevin equation as a function of reduced-field strength,  $\epsilon$ .

Now that we have a method for finding the average polarization given a field, the final piece is finding the field that is polarizing the dipoles. This is determined by calculating the superposition of all the solute generated fields at the solvent location  $\mathbf{r}$  given solute positions  $\mathbf{R}_i$ ,

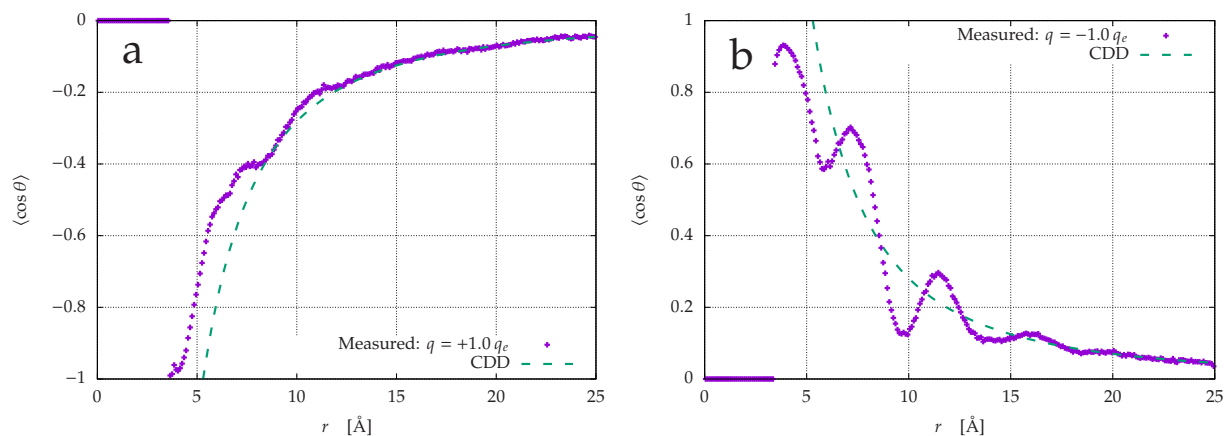
$$\epsilon_{\text{mf}}(\mathbf{r}) = \sum_{i=1}^N \epsilon_i(\mathbf{r} - \mathbf{R}_i) \quad (5.6)$$

where  $\epsilon_{\text{mf}}(\mathbf{r})$  is the reduced mean field felt by solvent particles at  $\mathbf{r}$ , and  $\epsilon_i(\mathbf{r} - \mathbf{R}_i)$  is the reduced-field at  $\mathbf{r}$  from the  $i^{\text{th}}$  solute at  $\mathbf{R}_i$ .

In practice we measure the polarization of the solvent around a charged solute particle and then solve for the reduced mean field that would yield that polarization using the inverse Langevin function approximated by,<sup>70</sup>

<sup>70</sup>Kröger, M. *Journal of Non-Newtonian Fluid Mechanics* **2015**, 223, 77–87.

$$\mathcal{L}^{-1}(x) \approx \frac{3x - x(6x^2 + x^4 - 2x^6)/5}{1 - x^2} \quad (5.7)$$



**Figure 5.11:** Mean polarization for the  $r_{\min} = 5$  Å model system as a function of solvent–solute distance,  $r$ , for the positive solute particle **(a)** and negative solute particle **(b)**. Both plots include the polarization prediction from assuming a constant density dielectric medium in dashed green.

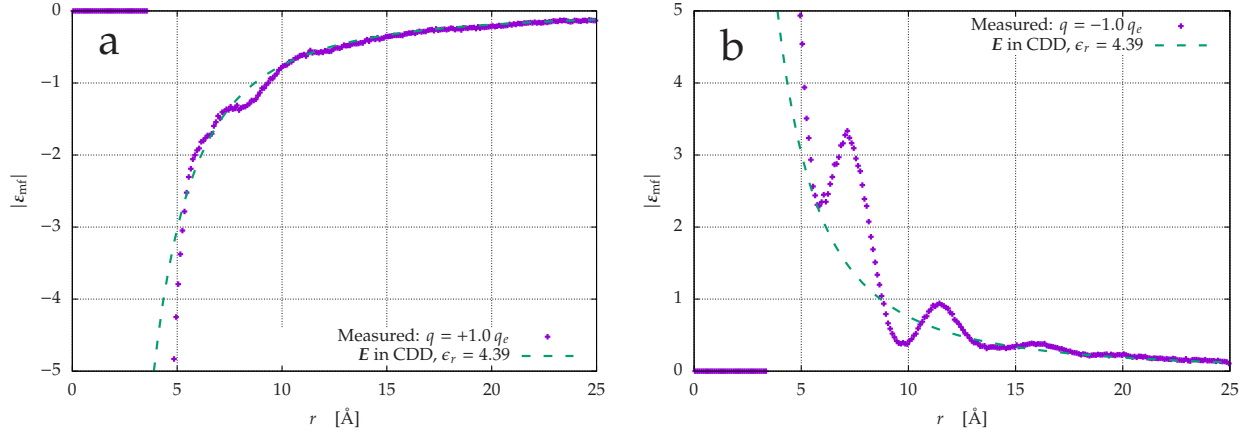
For the  $r_{\min} = 5$  Å solute model system the mean polarization for the positive and negative particles are shown in figure 5.11. The mean polarization has structure that oscillates about the theoretically predicted polarization of a constant density dielectric (CDD) medium. These oscillations and deviations are stronger for the negative solute particle but still present for both. The presence of these oscillations is something of a mystery because they occur in Stockmayer fluids that have their LJ space filling shape decoupled from their dipole direction.<sup>71</sup> While this is an interesting result, for our purposes here it is merely a set of data that can be used as input to generate the mean field as a function of distance,  $\epsilon_{\text{mf}}(r)$ , shown in figure 5.12.

### 5.3.2 Long Range Behavior

The structure that is present in both the mean polarization and the mean field at shorter distances, figures 5.11 and 5.12, will be captured in IS-SPA by cubic spline interpolation of the measured data. Coulombic forces however have significant magnitude at long range which means we need to approximate this behavior to make up for our measurements that only extend out to distance of 25 Å. We approximate this long range behavior by assuming that the field at large distances are correctly described by a CDD. The polarization density of a CDD goes linearly with the field via,

<sup>71</sup>Adams, D. J.; Rasaiah, J. C. *Faraday Discussions of the Chemical Society* **1977**, *64*, 22.





**Figure 5.12:** Reduced mean field for the  $r_{\min} = 5 \text{ \AA}$  system as a function of distance from the charged solute for the positive solute particle **(a)** and negative solute particle **(b)**. The mean field is calculated by solving equation (5.7) where  $x = \langle \cos \theta \rangle$ . Both plots include the mean field prediction from assuming a CDD medium in dashed green.

$$\mathbf{P} = \epsilon_0(\epsilon_r - 1)\mathbf{E} \quad (5.8)$$

where  $\epsilon_0$  is the vacuum permittivity,  $\epsilon_r$  is the dielectric constant of the medium, and  $\mathbf{P}$  is the polarization density. At large distances the field from a point charge in a CDD will be correctly described by,

$$\mathbf{E}(r) = \frac{q}{4\pi\epsilon_0\epsilon_r} \frac{1}{r^2} \hat{\mathbf{r}} \quad (5.9)$$

where  $q$  is the charge of the point charge generating the field, and  $r$  is the separation distance from the charge to the point in space being polarized. Also, the magnitude of the polarization density from equation (5.8) can be related to microscopic details via,

$$|\mathbf{P}| = \rho p \langle \cos \theta \rangle \quad (5.10)$$

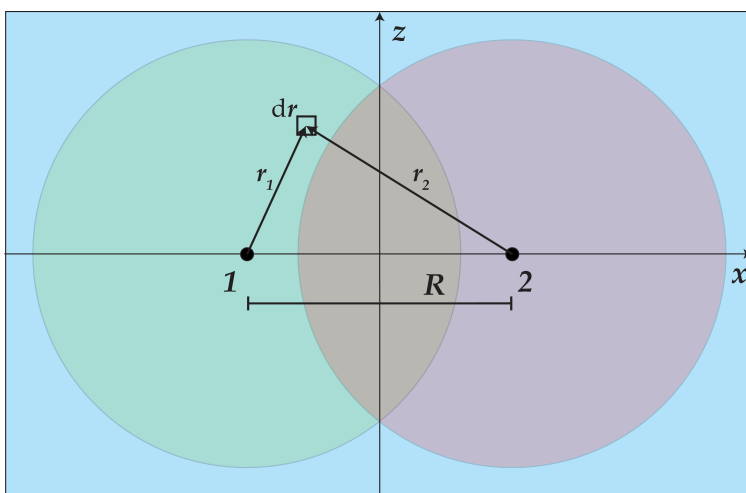
where  $\rho$  is the number density of the solvent,  $p$  is the static dipole moment of the solvent, and  $\langle \cos \theta \rangle$  is the average orientation of the dipole relative to the field. Putting everything together and solving for the orientation gives,

$$\langle \cos \theta \rangle = \frac{q}{4\pi\rho p} \left(1 - \frac{1}{\epsilon_r}\right) \frac{1}{r^2} \quad (5.11)$$

for the long range alignment of the solvent dipoles to the field from a solute charge. In all figures that contain relevant Coulombic results the CDD equivalent has been overlaid on the data to show the agreement at large distances between the theory and the measured data. Since there is good agreement between the measured data at large distances, ( $r > 18 \text{ \AA}$ ), the approximation of treating explicit chloroform as a CDD at large distances appears to be valid.

### 5.3.3 Application to Model System

To test the implementation of the dipole Coulombic force and the CDD approximation in IS-SPA we calculate the mean force and PMF as a function of solute–solute distance in our model system described in section 2.2.3. The LJ force is calculated in the same manner as was done previously by spline interpolating the LJ force histogram. The Coulombic force, because of its long range nature, is solved in a manner based on where the solvent lies with respect to the interaction volumes of the two solutes. Figure 5.13 shows the interaction volumes for the model system at a particular separation distance  $R$ .



**Figure 5.13:** Depiction of the interaction volumes in the model system for a particular separation distance  $R$ . The interaction volume of solute 1 is shown in green, solute 2 in red, shared volume in orange, and all other volume in blue.

The interaction volume refers to the spherical volume centered around a solute with a radius determined by the histogrammed input data, in this case the  $\epsilon_{mf}(r)$  histogram, which has measured data out to  $25 \text{ \AA}$  as seen in figure 5.12. Examining our mean field data, the measured data concur

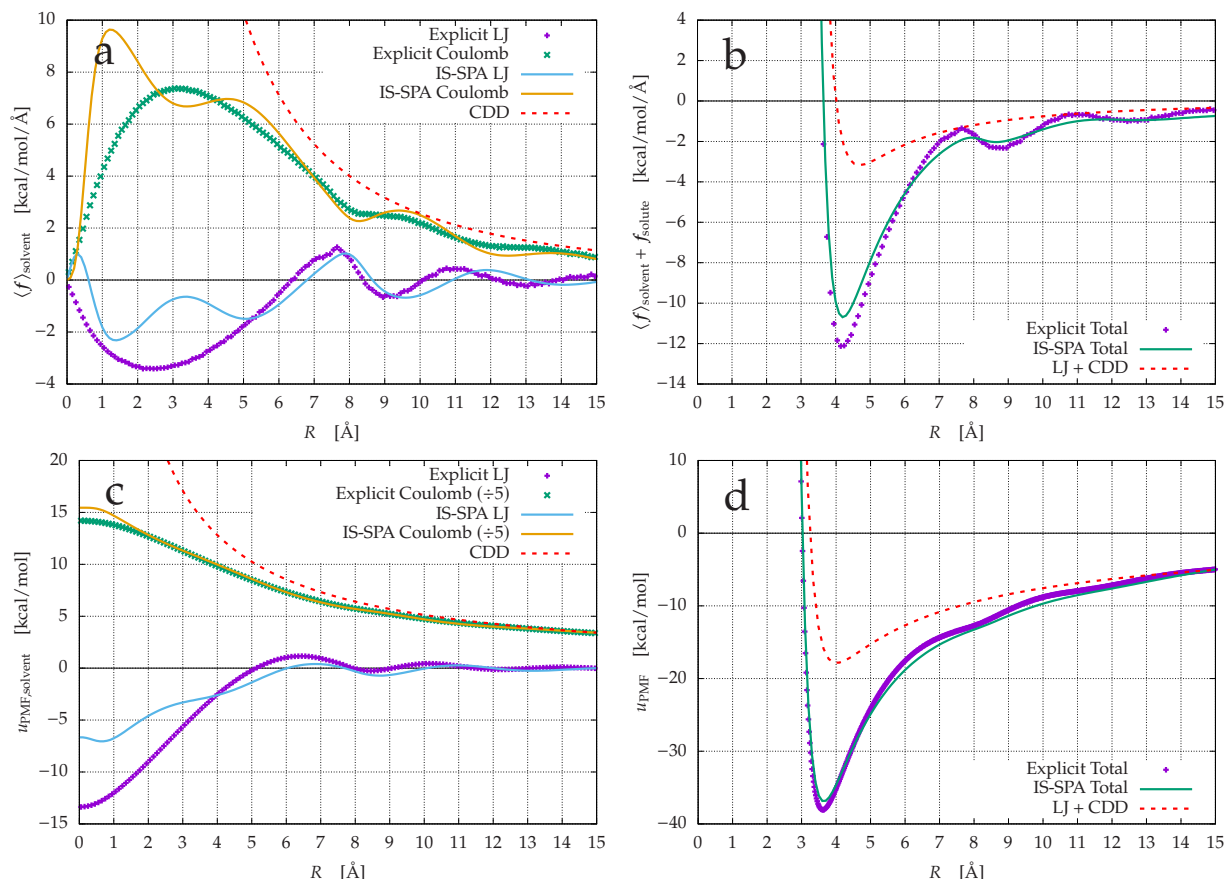
with the theoretical CDD prediction for both charges at all  $r \geq 18 \text{ \AA}$  so the interaction volume radii could be safely truncated to  $18 \text{ \AA}$  with CDD used for all distances greater than  $18 \text{ \AA}$ . For solvents that are within the interaction volumes of both solutes, (orange in fig. 5.13), the mean field is calculated by summing the interpolated values of the  $\epsilon_{mf}(r)$  histogram from each solute. For solvents within the interaction volume of only one solute, as the volume element  $d\mathbf{r}$  is in figure 5.13 for solute **1**, the mean field histogram is interpolated for solute **1** and the theoretical field in a CDD, eq. 5.9, is used to determine  $\epsilon_i(r - R_i)$  for solute **2**. The polarization is determined using the inverse Langevin function in equation (5.7) with the mean field as the argument. For all of the volume outside of the interaction volumes, (blue in fig. 5.13), an analytic correction term is used.

$$f_{\text{out}}(\mathbf{R}) = \begin{cases} -\frac{q_1 q_2}{4\pi\epsilon_0} \left(1 - \frac{1}{\epsilon_r}\right) \left[ \frac{R(8r_{\text{cut}} - 3R)}{24r_{\text{cut}}^4} - \frac{\log(1+R/r_{\text{cut}})}{8R^2} + \frac{r_{\text{cut}}}{8R(R+r_{\text{cut}})^2} - \frac{R^2}{32r_{\text{cut}}^4} + \frac{3}{16r_{\text{cut}}^2} \right] \hat{\mathbf{R}} & , R < 2r_{\text{cut}} \\ -\frac{q_1 q_2}{4\pi\epsilon_0} \left(1 - \frac{1}{\epsilon_r}\right) \left[ \frac{2}{3R^2} - \frac{1}{8R^2} \log\left(\frac{R+r_{\text{cut}}}{R-r_{\text{cut}}}\right) + \frac{r_{\text{cut}}(R^2+r_{\text{cut}}^2)}{4R(R^2-r_{\text{cut}}^2)} \right] \hat{\mathbf{R}} & , R > 2r_{\text{cut}} \end{cases} \quad (5.12)$$

where  $r_{\text{cut}}$  is the maximum distance of the measured data histogram.

The mean forces and PMFs of LJ and Coulomb interactions were calculated individually and plotted in figure 5.14 for  $r_{\text{min}} = 5 \text{ \AA}$  solutes with  $\pm 1.0 q_e$  charge. Panels **a** and **b** contain the individual parts and total of the mean force respectively, and panels **c** and **d** display the individual parts and total of the PMF respectively. The individual forces from IS-SPA follow the same trends as the explicit ones. They only deviate minimally and are often times opposite in sign such that the error from the LJ force cancels with the Coulomb force error. The resultant cancellations end up smoothing out the total force compared to the explicit which has appreciable oscillations at distances larger than  $7 \text{ \AA}$ , despite this difference IS-SPA still captures the overall shape of the well remarkably well. The solvent forces, figure 5.14a, also follow the expected behavior of the LJ force favoring dimerization and the Coulombic force destabilizing dimerization. Since the system is under external pressure all of the solvent molecules on average are pushing in on the solutes, thus any reduction in volume that the solutes can undergo by dimerizing will be favored by the LJ solvent force. Solvent Coulombic forces in general rip apart dimers because the alignment of solvent dipoles along the solute dimer field put forces on the dimer atoms that repel them from each other. The PMFs are perhaps even more compelling with many of the deviations of the force

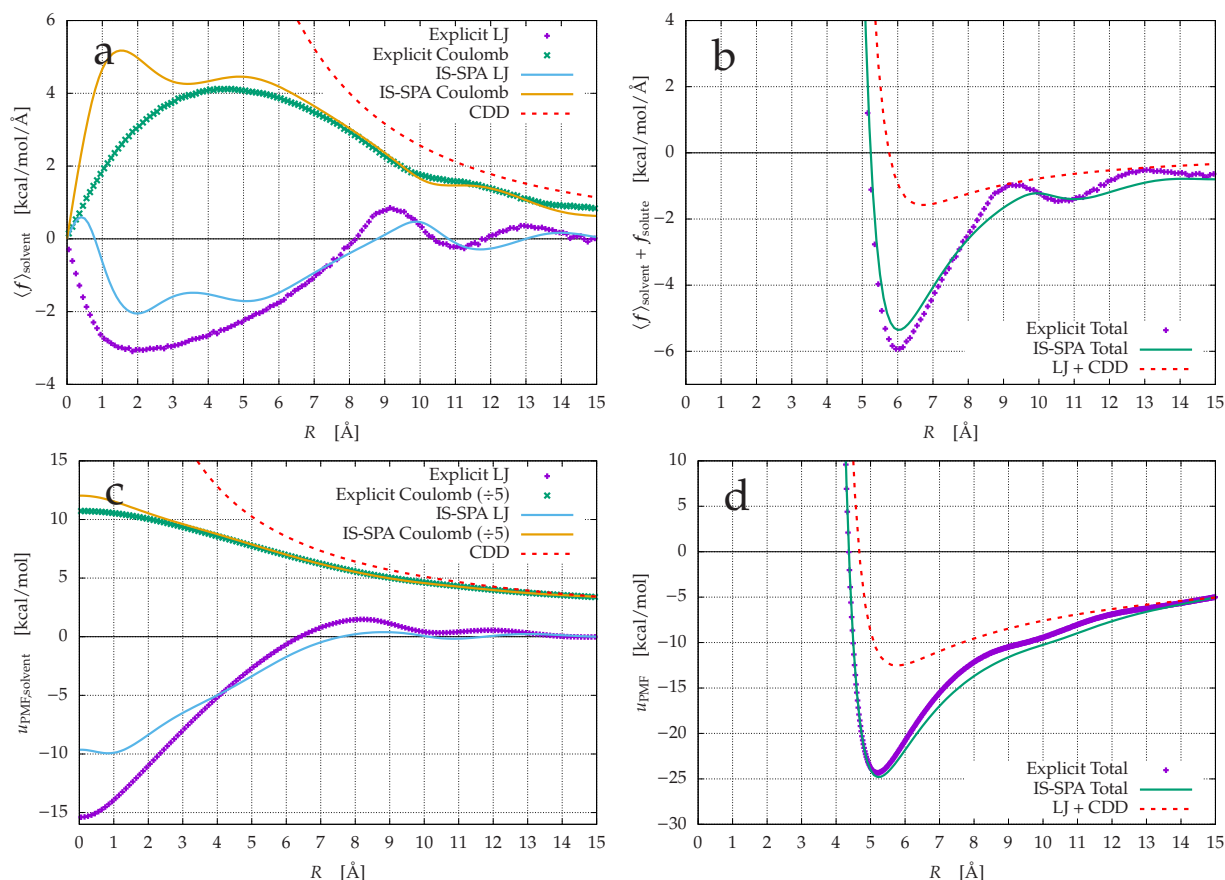
being smoothed out by integration. The final result of which is the total PMF in panel 5.14d where the explicit and IS-SPA curves are nearly identical.



**Figure 5.14:** Model system mean force and PMF results of IS-SPA with dipole Coulombic forces as a function of solute-solute distance  $R$  compared to explicit MD for  $r_{\min} = 5 \text{ \AA}$  and  $q = \pm 1.0 q_e$  solute particles. **(a)** Mean force for the LJ and Coulombic portions separately. **(b)** Total force with direct solute-solute term. **(c)** PMFs of LJ and Coulombic portions separately. **(d)** Total PMF with direct solute-solute term. Each panel also includes a dashed red curve that is the Coulombic force or PMF assuming the solvent is a CDD.

Figures 5.14a,b also include the force assuming the two solute charges are in CDD, and figures 5.14c,d show the total PMF of the CDD potential energy. In the panels with the total forces and PMFs (fig. 5.14b,d) the dashed LJ and CDD curve demonstrates how modeling the solvent solely as a CDD would perform. As is evidenced by the depth of the well the CDD under-stabilizes the contact pair of the solutes for two reasons: it does not account for the attractive LJ force from the solvent seen in figure 5.14a at contact distances  $r \leq 6.5 \text{ \AA}$ , and it over predicts the Coulomb screening at those same distances. This originates from the fact that the solvent itself has volume

exclusion to it such that the solvent dipole can only get so close to the solute. Also, because of the finite size of the solvent it packs into shells of enhanced and diminished density which give both the Van der Waals and electrostatic components structure that CDD by definition lacks. Evidently these two qualities are important for reproducing the explicit mean force and PMF because the results of using solely CDD are do not agree with the explicit results.



**Figure 5.15:** Model system mean force and PMF results of IS-SPA with dipole Coulombic forces as a function of solute-solute distance  $R$  compared to explicit MD for  $r_{\min} = 7 \text{ \AA}$  and  $q = \pm 1.0 q_e$  solute particles. **(a)** Mean force for the LJ and Coulombic portions separately. **(b)** Total force with direct solute-solute term. **(c)** PMFs of LJ and Coulombic portions separately. **(d)** Total PMF with direct solute-solute term. Each panel also includes a dashed red curve that is the Coulombic force or PMF assuming the solvent is a CDD.

The equivalent figure to 5.14 for  $r_{\min} = 7 \text{ \AA}$  solutes is shown in figure 5.15. The same trends in mean force and PMF persist for the charged  $r_{\min} = 7 \text{ \AA}$  solute, with smoothed out oscillations compared to the explicit but similar overall shape. It is somewhat surprising that the results are not more accurate considering the marked improvement of the uncharged model system. Further

study of the effect of solute particle size would be interesting for the charged solutes as well as the uncharged solutes because of the somewhat inconsistent results present at this point.

Overall the approximations of chloroform as a sphere in LJ potential and as a dipole in electrostatic potential appear to be valid approximations that will lead to an implicit model for chloroform with a promising degree of accuracy and efficiency compared to AESMD simulations. The next step to test this theory will be to generate the difference vector plots (fig. 5.9) using the dipole IS-SPA theory presented here. Finally, the LO PMF will be generated and compared to the explicit PMF, figure 1.3.

## Chapter 6

### Future Work

Firstly, to wrap up the results and story presented in chapter 5, a powerful figure to generate would be the force difference vector figure for LO with dipole Coulombic forces and compare to the monopolar version, figure 5.9. If our hypothesis was correct the Coulombic force difference vectors that were circled should be diminished in intensity and perhaps not pointing in opposite directions any more. The pointing of the vectors will depend on how well the dipole approximation matches the explicit solvent dipole forces as well as how much other terms of the electrostatic force are effecting the solute atoms. Finally, the PMF of LO from IS-SPA simulation should be calculated and compared to the AESMD PMF, figure 1.3.

With the results shown in chapter 5 we are very near to having a full story for a publication that includes both the development for chloroform and the dipole additions to IS-SPA. Section 6.1 outlines a further development that could improve the accuracy of the Coulombic dipole approximation of IS-SPA if necessary in the future. Section 6.2 outlines another expansion to IS-SPA that is focused on solvent molecules that are flexible and/or solvent molecules that are composed of distinctly different parts such as a polar group bonded to a non-polar.

#### 6.1 Dynamic Dipole in IS-SPA

In the event of needing further refinement to the Coulombic force in IS-SPA I think one approximation to refine would be the static dipole approximation. Previously we assumed that the dipole magnitude,  $p$ , was static and solved for the alignment of the dipole,  $\langle \cos \theta \rangle$ , as a function of solvent–solute displacement,  $r$ , and arrived at equation (5.5). If we instead solve for the alignment of the dipole when the magnitude is allowed to fluctuate in a Gaussian distribution about a mean,  $p_0$ , we go from equation (6.1) to (6.2),

$$\mathcal{P}(p, \cos \theta) \propto e^{E \cdot p / T} \delta(p - p_0) p^2 dp \, d\cos \theta \quad (6.1)$$

$$\propto e^{E \cdot p / T} e^{-(p-p_0)^2 / 2\sigma^2} p^2 dp \, d\cos \theta \quad (6.2)$$

where  $p_0$  is the mean dipole magnitude, and  $\sigma^2$  is the variance of the fluctuations. As before,  $E$  is the external electric field vector,  $\mathbf{p}$  is the dipole moment vector,  $T$  is the temperature in units of energy, and  $p$  is the dipole magnitude variable. The delta-function in equation (6.1) which represents the dipole being fixed at a magnitude of  $p_0$  is replaced in equation (6.2) by a Gaussian distribution centered at the mean dipole magnitude,  $p_0$ .

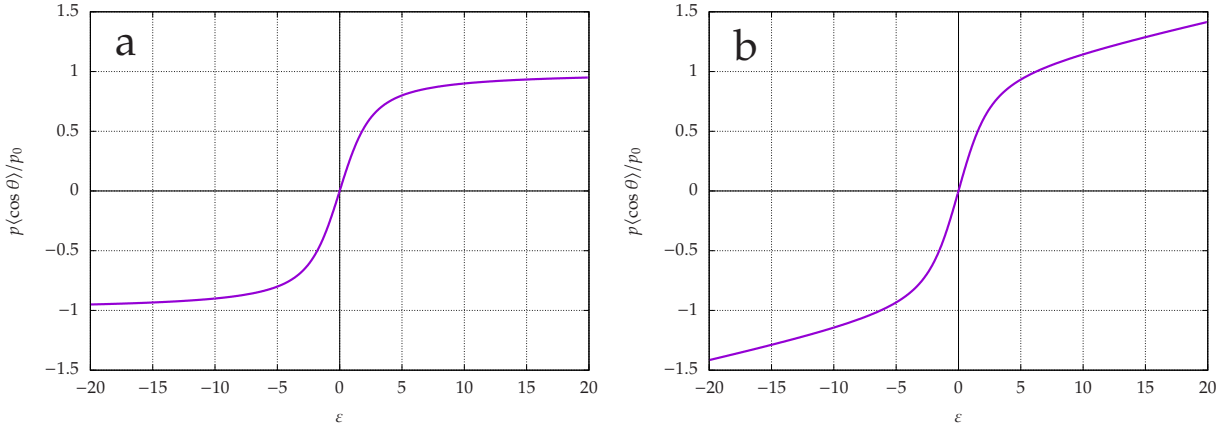
This time instead of solving for only the average alignment  $\langle \cos \theta \rangle$  because we were using the approximation  $p = p_0$ , we solve for  $p \langle \cos \theta \rangle$  which is the average dipole moment.

$$p \langle \cos \theta \rangle = \frac{\int_0^{\infty} p^2 dp \int_{-1}^1 d\cos \theta p \cos \theta \mathcal{P}(p, \cos \theta)}{\int_0^{\infty} p^2 dp \int_{-1}^1 d\cos \theta \mathcal{P}(p, \cos \theta)} \quad (6.3)$$

which eventually yields,

$$\frac{p \langle \cos \theta \rangle}{p_0} = \frac{(2\varepsilon^2 \zeta^2 - 1) \sinh \varepsilon + \varepsilon(1 + \varepsilon^2 \zeta^4) \cosh \varepsilon}{2\varepsilon(\sinh \varepsilon + \varepsilon \zeta^2 \cosh \varepsilon)} \quad (6.4)$$

where  $\varepsilon$  is the reduced-field, ( $\varepsilon = p_0 E/T$ ), and  $\zeta$  is the fractional variance, ( $\zeta^2 = \sigma^2/p_0^2$ ).



**Figure 6.1:**  $p \langle \cos \theta \rangle / p_0$ , equation (6.4), as a function of reduced field  $\varepsilon$ . **(a)** For fractional variance  $\zeta = 0$ , same behavior as a static dipole (fig. 5.10), asymptotically approaching complete polarization,  $\langle \cos \theta \rangle = 1$ . **(b)** For  $\zeta > 0$  the behavior at large  $\varepsilon$  is linear in  $\varepsilon$  with slope of  $\zeta^2/2$ .

The behavior of the dynamic dipole, equation (6.4), converges to static dipole behavior, equation 5.10, when  $\zeta = 0$ , figure 6.1a, which is consistent. When  $\zeta > 0$ , (fig. 6.1b) i.e. the dipole magnitude is allowed to fluctuate, the polarization follows the same sigmoidal behavior as the



Langevin function for small  $\varepsilon$  but for large  $\varepsilon$  the polarization is linear in  $\varepsilon$  with a slope of  $\zeta^2/2$ . In other words, the dynamic dipole behaves like a static dipole at small field strengths, aligning its dipole along the field, but can continue to be polarized at large fields by stretching the dipole once it is already aligned.

Another consideration for the dipole expansion to IS-SPA is the location of the dipole within the solvent's excluded volume. The offset parameter  $d$  did not make much difference on either the LJ or the Coulombic force data but perhaps for other solvents it may end up being important. My thought is that perhaps it would be important to mark the location of the dipole inside the solvent molecule. If for example the solvent has a large excluded volume with a dipole on one side of the volume, perhaps it would be important then to keep track of where the dipole is located with an offset distance  $d$ .

## 6.2 IS-SPA Applied to Flexible Solvent Molecules

In my independent research proposal I outlined a method for applying IS-SPA to flexible solvents. This section is a brief outline of the core idea of that proposal.

So far IS-SPA has been used to model both water and chloroform by simplifying the solvent to spherically symmetric particles where the only degree of freedom is the distance between solvent and solute. For more complicated solvents that require orientational information, other degrees of freedom would need to be sampled. For example, one for the position and another for the orientation, assuming only one orientational degree of freedom is sufficient. The more degrees of freedom required to place a solvent particle the more expensive the algorithm.

For inflexible solvents such as water and chloroform, the internal degrees of freedom can be ignored. A more complicated version of modeling these solvents in IS-SPA would be to define the distribution and force functions,  $g_i(|\mathbf{r} - \mathbf{R}_i|)$  and  $f_{i,\text{solv}}(\mathbf{r} - \mathbf{R}_i)$ , as 3-dimensional functions of separation distance  $|\mathbf{r} - \mathbf{R}_i|$ , polar tilt angle  $\theta$ , and azimuthal twist angle  $\phi$ , with respect to the solutes:  $g_i(|\mathbf{r} - \mathbf{R}_i|, \theta, \phi)$  and  $f_{i,\text{solv}}(|\mathbf{r} - \mathbf{R}_i|, \theta, \phi)$ . Even though using 3-dimensional functions would be more complicated than the previously mentioned 1-dimensional functions, the Monte Carlo sampling situation is still tractable for the fact that the solvents are assumed to have completely fixed internal degrees of freedom. However, if the solvent molecule is flexible and can sample an array of intramolecular conformations of its own, such as alkanes like pentane or hexane, the

process of placing a solvent in the Monte Carlo integration becomes more complicated. Pentane is an ideal model solvent for studying this expansion to IS-SPA. It lacks appreciable charge on any of its atoms thus requiring only LJ force considerations. It has enough atoms to be flexible and has more than one stable conformation necessitating the expansion of IS-SPA. Butane would also be sufficient except that it is gaseous at STP whereas pentane is a liquid.

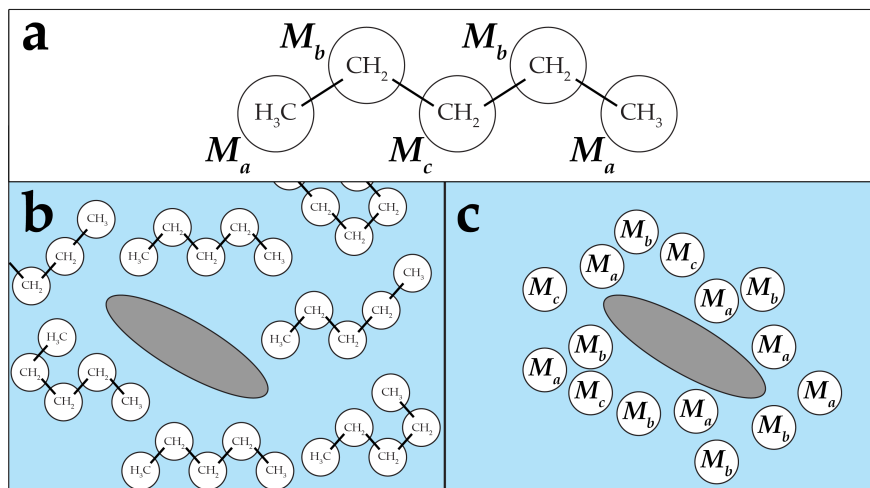
One method of simplifying pentane from a physical perspective is modeling each pentane molecule as a bonded chain of five unified atoms similar to the transferable potentials for phase equilibria (TraPPE)<sup>72</sup> model which has unique force fields for CH<sub>2</sub> and CH<sub>3</sub> groups respectively. However, unified atom approaches like TraPPE still require the computation of large numbers of degrees of freedom like AESMD simulations, the number of solvent atoms is just decreased by a scalar factor equal to the average number of atoms in the unified atoms.

Simplifying pentane from the perspective of IS-SPA would be to model each pentane as a single spherical or perhaps ellipsoidal particle. However since pentane is flexible the IS-SPA approach becomes complicated quickly because there is the question of how to sample the internal degrees of freedom of the solvent. One option is to place the methyl groups of pentane one-by-one by sampling the intramolecular distribution function and model each methyl group as a single particle analogous to water and chloroform in previous IS-SPA development. Another is to place each pentane preconfigured in its most likely configurations, as measured from MD simulation as a function of dihedral angle, to avoid having to sample an intramolecular distribution function in addition to the intermolecular distribution function already being sampled. A third option, and the focus of this section, is the methyl-atom strategy detailed below.

### 6.2.1 Methyl-Atom Strategy

The methyl-atom strategy uses the mechanics, approximations, and efficiency of 1-dimensional spherical IS-SPA but places individual coarse-grained methyl groups, *methyl-atoms*, instead of placing entire pentane molecules. In pentane there are three symmetrically unique methyl-groups as shown in figure 6.2a: the terminal CH<sub>3</sub> groups, the CH<sub>2</sub> groups bound to the terminal groups, and the middle CH<sub>2</sub> group represented by the methyl-atom types  $M_a$ ,  $M_b$ , and  $M_c$  respectively.

Each methyl-atom type has a unique distribution and force function as measured from AESMD simulation data, figure 6.3. These data suggest the methyl groups of pentane can be coarse-grained

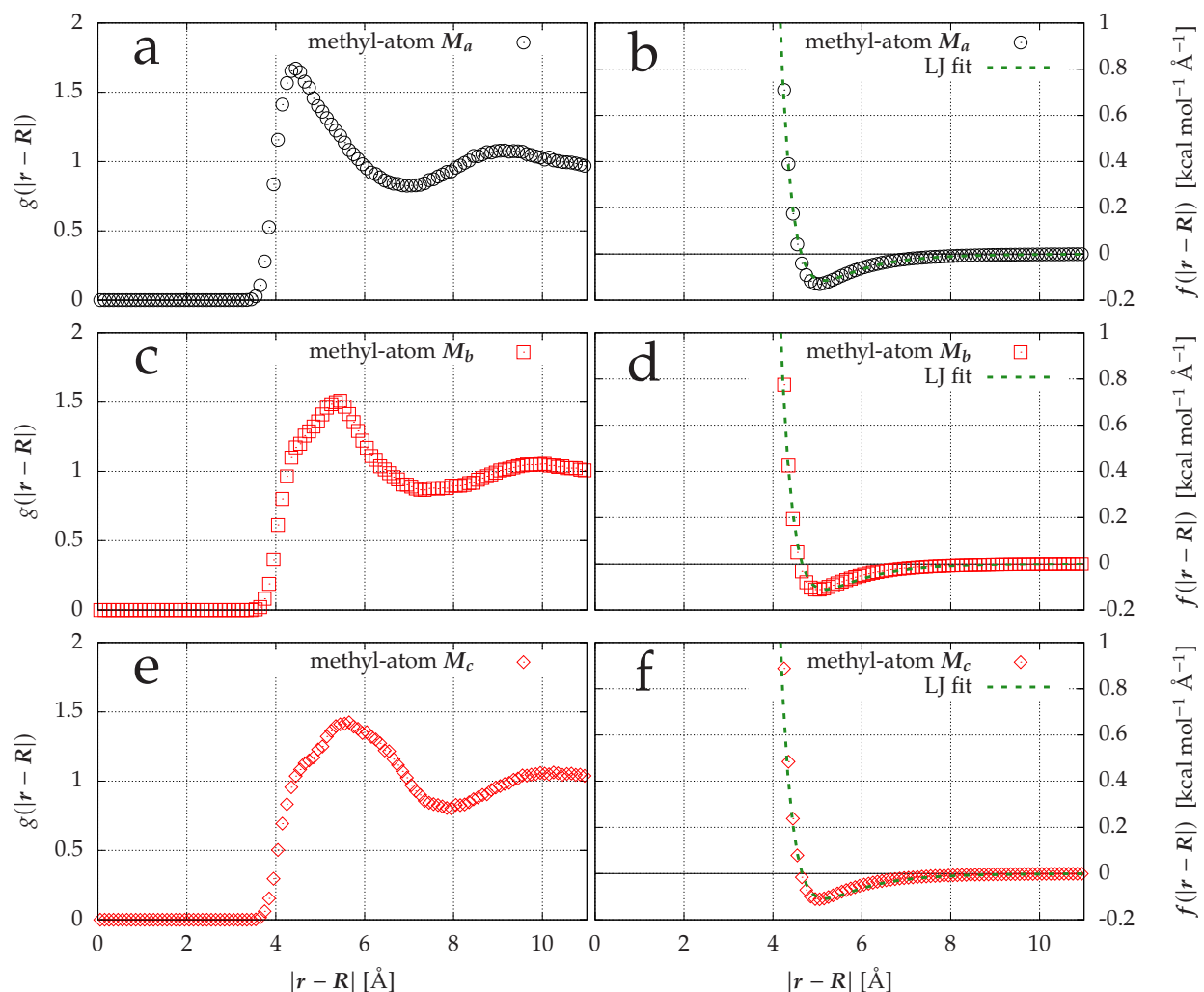


**Figure 6.2:** (a) Pentane molecule with the symmetrically unique methyl-atoms labeled  $M_a$ ,  $M_b$ , and  $M_c$ . (b) Solute solvated by explicit pentane coarse-grained into a chain of five unified methyl group particles. (c) Solute solvated by methyl-atoms randomly placed around the solute for the Monte Carlo integration in IS-SPA.

into spherically symmetric methyl-atom particles, making the Monte Carlo sampling 1-dimensional and thus more efficient than placing and orienting explicit methyl groups.

In this strategy the individual methyl-atoms are placed around the solute for the Monte Carlo integration without correlation to what other methyl-atoms are in the vicinity i.e. they are not bound to each other as depicted in figure 6.2b but instead are completely independent particles as in figure 6.2c. In this respect this strategy is unphysical because in explicit pentane the presence of a terminal  $\text{CH}_3$  group would mean a  $\text{CH}_2$  group would have to be near by since they are bound, but with methyl-atoms there could be a  $M_a$  group without a  $M_b$  group near it. However, as long as the mean force on the solutes is accurately captured the PMF will be captured as well. Since there is this potential for unphysical results it may be important to develop an algorithm for deciding whether a methyl-atom can be physically placed in a given volume where an entire pentane molecule might not fit. Also it is a likely complication that a ratio of the types of methyl-atoms in pentane will need to be satisfied during Monte Carlo sampling; two terminal  $M_a$  groups to two  $M_b$  groups to one middle  $M_c$  type methyl-atoms in a pentane molecule.

The explicit data measured from MD simulation, (*points*) in figure 6.3b,d,f, are calculated by projecting the sum of the forces from each atom of a particular methyl group onto the separation vector of the carbon atom and solute. The result is an orientationally averaged force from that



**Figure 6.3:** Symmetrically unique methyl-atom radial distribution functions (a,c,e) and forces (b,d,f) of pentane, black points correspond to CH<sub>3</sub> groups and red points to CH<sub>2</sub> groups. The forces are fit to single LJ force functions.

methyl group i.e. a coarse-grained spherically symmetric methyl group force. The distribution functions, figure 6.3a,c,e, are measured using the carbon position.

Fitting to single LJ force functions shown in figure 6.3b,d,f and table 6.1 demonstrate that there are discrepancies between the explicit methyl-atom force and a single fitted LJ force. Each methyl-atom also has a unique distribution function as seen in figure 6.3a,c,e which would be used for weighting the respective methyl-atoms.

Adding flexible solvent implicitization capability to IS-SPA opens up possibilities of more types of systems for large scale molecular simulations that would have been previously unfeasible.

**Table 6.1:** The LJ parameters from the fits shown in figure 6.3b,d,f for the LJ force functional form:  $f_{LJ}(r) = -4\epsilon \left[ 12 \left( \frac{\sigma}{r} \right)^{13} - 6 \left( \frac{\sigma}{r} \right)^7 \right]$ .

Methyl-atom	$\epsilon$ [kcal/mol]	$\sigma$ [Å]
$M_a$ (CH <sub>3</sub> )	0.198639	4.13053
$M_b$ (CH <sub>2</sub> )	0.193287	4.14888
$M_c$ (CH <sub>2</sub> )	0.188080	4.14888

## Bibliography

1. Baker, N. A. *Current Opinion in Structural Biology* **2005**, *15*, 137–143 (cit. on p. 2).
2. Onufriev, A.; Bashford, D.; Case, D. A. *The Journal of Physical Chemistry B* **2000**, *104*, 3712–3720 (cit. on p. 2).
3. Chandler, D.; Andersen, H. C. *The Journal of Chemical Physics* **1972**, *57*, 1930–1937 (cit. on p. 2).
4. Beglov, D.; Roux, B. *The Journal of Chemical Physics* **1996**, *104*, 8678–8689 (cit. on p. 2).
5. Mattson, M. A.; Green, T. D.; Lake, P. T.; McCullagh, M.; Krummel, A. T. *The Journal of Physical Chemistry B* **2018**, *122*, 4891–4900 (cit. on pp. 4, 56, 57, 59, 60, 62).
6. Lake, P. T.; McCullagh, M. *Journal of Chemical Theory and Computation* **2017** (cit. on pp. 5, 6, 8, 10).
7. Case, D.; Cerutti, D.; Cheatham, T.; Darden, T.; Duke, R.; Giese, T.; Gohlke, H.; Goetz, A.; Greene, D.; Homeyer, N, et al. AMBER16 Package (cit. on pp. 5, 19, 103).
8. Sugita, Y.; Okamoto, Y. *Chemical Physics Letters* **1999**, *314*, 141–151 (cit. on p. 6).
9. Sugita, Y.; Kitao, A.; Okamoto, Y. *The Journal of chemical physics* **2000**, *113*, 6042–6051 (cit. on p. 6).
10. Fukunishi, H.; Watanabe, O.; Takada, S. *The Journal of Chemical Physics* **2002**, *116*, 9058–9067 (cit. on p. 6).
11. Kirkwood, J. G. *The Journal of Chemical Physics* **1935**, *3*, 300–313 (cit. on p. 9).
12. Jorgensen, W. L.; Chandrasekhar, J.; Madura, J. D.; Impey, R. W.; Klein, M. L. *The Journal of Chemical Physics* **1983**, *79*, 926–935 (cit. on pp. 10, 69).
13. Press, W. H.; Teukolsky, S. A.; Vetterling, W. T.; Flannery, B. P., *Numerical Recipes in Fortran 77: The Art of Scientific Computing*, Second; Cambridge England: 1996; Vol. 1 (cit. on pp. 16, 17).
14. Cieplak, P.; Caldwell, J.; Kollman, P. *Journal of Computational Chemistry* **2001**, *22*, 1048–1057 (cit. on pp. 19, 21, 69).
15. Wang, J.; Wolf, R. M.; Caldwell, J. W.; Kollman, P. A.; Case, D. A. *Journal of Computational Chemistry* **2004**, *25*, 1157–1174 (cit. on pp. 20, 39, 59).

16. Tang, C. W. *Applied Physics Letters* **1986**, *48*, 183–185 (cit. on p. 36).
17. Zhang, X.; Lu, Z.; Ye, L.; Zhan, C.; Hou, J.; Zhang, S.; Jiang, B.; Zhao, Y.; Huang, J.; Zhang, S.; Liu, Y.; Shi, Q.; Liu, Y.; Yao, J. *Advanced Materials* **2013**, *25*, 5791–5797 (cit. on p. 36).
18. Zhou, E.; Cong, J.; Wei, Q.; Tajima, K.; Yang, C.; Hashimoto, K. *Angewandte Chemie International Edition* **2011**, *50*, 2799–2803 (cit. on p. 36).
19. Nielsen, C. B.; Holliday, S.; Chen, H.-Y.; Cryer, S. J.; McCulloch, I. *Accounts of Chemical Research* **2015**, *48*, 2803–2812 (cit. on p. 36).
20. Zang, L.; Che, Y.; Moore, J. S. *Accounts of Chemical Research* **2008**, *41*, 1596–1608 (cit. on p. 36).
21. Chen, S.; Slattum, P.; Wang, C.; Zang, L. *Chemical Reviews* **2015**, *115*, 11967–11998 (cit. on p. 36).
22. Würthner, F.; Chen, Z.; Dehm, V.; Stepanenko, V. *Chemical Communications* **2006**, 1188–1190 (cit. on p. 36).
23. Keivanidis, P. E.; Howard, I. A.; Friend, R. H. *Advanced Functional Materials* **2008**, *18*, 3189–3202 (cit. on p. 36).
24. Mäkinen, A. J.; Melnyk, A. R.; Schoemann, S.; Headrick, R. L.; Gao, Y. *Physical Review B* **1999**, *60*, 14683–14687 (cit. on p. 36).
25. Russ, B.; Robb, M. J.; Brunetti, F. G.; Miller, P. L.; Perry, E. E.; Patel, S. N.; Ho, V.; Chang, W. B.; Urban, J. J.; Chabynyc, M. L.; Hawker, C. J.; Segalman, R. A. *Advanced Materials* **2014**, *26*, 3473–3477 (cit. on p. 36).
26. Ye, T.; Singh, R.; Butt, H.-J.; Floudas, G.; Keivanidis, P. E. *ACS Applied Materials & Interfaces* **2013**, *5*, 11844–11857 (cit. on p. 36).
27. Balakrishnan, K.; Datar, A.; Naddo, T.; Huang, J.; Oitker, R.; Yen, M.; Zhao, J.; Zang, L. *Journal of the American Chemical Society* **2006**, *128*, 7390–7398 (cit. on pp. 36, 46).
28. Shao, C.; Grüne, M.; Stolte, M.; Würthner, F. *Chemistry – A European Journal* **2012**, *18*, 13665–13677 (cit. on pp. 36, 37, 46).
29. Fennel, F.; Wolter, S.; Xie, Z.; Plötz, P.-A.; Kühn, O.; Würthner, F.; Lochbrunner, S. *Journal of the American Chemical Society* **2013**, *135*, 18722–18725 (cit. on pp. 36, 37).

30. Kaiser, T. E.; Wang, H.; Stepanenko, V.; Würthner, F. *Angewandte Chemie International Edition* **2007**, *46*, 5541–5544 (cit. on pp. 36, 37).
31. Samanta, S.; Chaudhuri, D. *The Journal of Physical Chemistry Letters* **2017**, *8*, 3427–3432 (cit. on p. 37).
32. Zanni, M. T.; Gnanakaran, S.; Stenger, J.; Hochstrasser, R. M. *The Journal of Physical Chemistry B* **2001**, *105*, 6520–6535 (cit. on p. 37).
33. Wang, L.; Middleton, C. T.; Singh, S.; Reddy, A. S.; Woys, A. M.; Strasfeld, D. B.; Marek, P.; Raleigh, D. P.; de Pablo, J. J.; Zanni, M. T.; Skinner, J. L. *Journal of the American Chemical Society* **2011**, *133*, 16062–16071 (cit. on p. 37).
34. Moran, S. D.; Zanni, M. T. *The Journal of Physical Chemistry Letters* **2014**, *5*, 1984–1993 (cit. on p. 37).
35. Ganim, Z.; Tokmakoff, A. *Biophysical Journal* **2006**, *91*, 2636–2646 (cit. on p. 37).
36. Krummel, A. T.; Mukherjee, P.; Zanni, M. T. *The Journal of Physical Chemistry B* **2003**, *107*, 9165–9169 (cit. on p. 37).
37. Peng, C. S.; Jones, K. C.; Tokmakoff, A. *Journal of the American Chemical Society* **2011**, *133*, 15650–15660 (cit. on p. 37).
38. Anna, J. M.; King, J. T.; Kubarych, K. J. *Inorganic Chemistry* **2011**, *50*, 9273–9283 (cit. on p. 37).
39. Oudenhoven, T. A.; Joo, Y.; Laaser, J. E.; Gopalan, P.; Zanni, M. T. *The Journal of Chemical Physics* **2015**, *142*, 212449 (cit. on p. 37).
40. Cyran, J. D.; Krummel, A. T. *The Journal of Chemical Physics* **2015**, *142*, 212435 (cit. on pp. 37, 56, 58).
41. Cyran, J. D.; Nite, J. M.; Krummel, A. T. *The Journal of Physical Chemistry B* **2015**, *119*, 8917–8925 (cit. on pp. 38, 56, 58).
42. Phillips, J. C.; Braun, R.; Wang, W.; Gumbart, J.; Tajkhorshid, E.; Villa, E.; Chipot, C.; Skeel, R. D.; Kalé, L.; Schulten, K. *Journal of Computational Chemistry* **2005**, *26*, 1781–1802 (cit. on pp. 39, 47, 59).



43. Bayly, C. I.; Cieplak, P.; Cornell, W.; Kollman, P. A. *The Journal of Physical Chemistry* **1993**, *97*, 10269–10280 (cit. on pp. 39, 59).
44. Fox, T.; Kollman, P. A. *The Journal of Physical Chemistry B* **1998**, *102*, 8070–8079 (cit. on pp. 39, 59).
45. Kumar, S.; Rosenberg, J. M.; Bouzida, D.; Swendsen, R. H.; Kollman, P. A. *Journal of Computational Chemistry* **1992**, *13*, 1011–1021 (cit. on pp. 40, 59).
46. Frisch, M.; Trucks, G.; Schlegel, H.; Scuseria, G.; Robb, M.; Cheeseman, J.; Scalmani, G.; Barone, V.; Mennucci, B.; Petersson, G., et al. *Gaussian Inc, Wallingford CT, 2013* **2013** (cit. on p. 40).
47. Ponnuswamy, N.; Stefankiewicz, A. R.; Sanders, J. K. M.; Pantoş, G. D. In *Constitutional Dynamic Chemistry*, Barboiu, M., Ed.; Topics in Current Chemistry; Springer: Berlin, Heidelberg, 2012, pp 217–260 (cit. on p. 43).
48. Tambara, K.; Olsen, J.-C.; E. Hansen, D.; Dan Pantoş, G. *Organic & Biomolecular Chemistry* **2014**, *12*, 607–614 (cit. on p. 43).
49. Sukul, P. K.; Asthana, D.; Mukhopadhyay, P.; Summa, D.; Muccioli, L.; Zannoni, C.; Beljonne, D.; Rowan, A. E.; Malik, S. *Chemical Communications* **2011**, *47*, 11858–11860 (cit. on p. 46).
50. Krummel, A. T.; Zanni, M. T. *The Journal of Physical Chemistry B* **2006**, *110*, 13991–14000 (cit. on pp. 56, 57).
51. Sanstead, P. J.; Tokmakoff, A. *The Journal of Physical Chemistry B* **2018**, *122*, 3088–3100 (cit. on p. 56).
52. Ganim, Z.; Chung, H. S.; Smith, A. W.; DeFlores, L. P.; Jones, K. C.; Tokmakoff, A. *Accounts of Chemical Research* **2008**, *41*, 432–441 (cit. on pp. 56, 57).
53. Ghosh, A.; Ostrander, J. S.; Zanni, M. T. *Chemical Reviews* **2017**, *117*, 10726–10759 (cit. on p. 56).
54. Choi, H.; Paek, S.; Song, J.; Kim, C.; Cho, N.; Ko, J. *Chemical Communications* **2011**, *47*, 5509–5511 (cit. on p. 56).
55. Díaz-García, M. A.; Calzado, E. M.; Villalvilla, J. M.; Boj, P. G.; Quintana, J. A.; Céspedes-Guirao, F. J.; Fernández-Lázaro, F.; Sastre-Santos, Á. *Synthetic Metals* **2009**, *159*, 2293–2295 (cit. on p. 56).

56. Schmidt-Mende, L.; Fechtenkötter, A.; Müllen, K.; Moons, E.; Friend, R. H.; MacKenzie, J. D. *Science* **2001**, *293*, 1119–1122 (cit. on p. 56).
57. Shetty, A. S.; Zhang, J.; Moore, J. S. *Journal of the American Chemical Society* **1996**, *118*, 1019–1027 (cit. on p. 56).
58. Martinez, C. R.; Iverson, B. L. *Chemical Science* **2012**, *3*, 2191–2201 (cit. on p. 56).
59. Waters, M. L. *Current Opinion in Chemical Biology* **2002**, *6*, 736–741 (cit. on p. 56).
60. Shi, M.-M.; Chen, Y.; Nan, Y.-X.; Ling, J.; Zuo, L.-J.; Qiu, W.-M.; Wang, M.; Chen, H.-Z. *The Journal of Physical Chemistry B* **2011**, *115*, 618–623 (cit. on p. 56).
61. Peng, C. S.; Fedeles, B. I.; Singh, V.; Li, D.; Amariuta, T.; Essigmann, J. M.; Tokmakoff, A. *Proceedings of the National Academy of Sciences* **2015**, *112*, 3229–3234 (cit. on p. 57).
62. Fayer, M. D.; Moilanen, D. E.; Wong, D.; Rosenfeld, D. E.; Fenn, E. E.; Park, S. *Accounts of Chemical Research* **2009**, *42*, 1210–1219 (cit. on p. 57).
63. Volkov, V.; Hamm, P. *Biophysical Journal* **2004**, *87*, 4213–4225 (cit. on p. 57).
64. Tan, H.-S.; Warren, W. S. *Optics Express* **2003**, *11*, 1021–1028 (cit. on p. 58).
65. Kuhs, C. T.; Luther, B. M.; Krummel, A. T. *IEEE Journal of Selected Topics in Quantum Electronics* **2019**, *25*, 1–13 (cit. on p. 58).
66. Shim, S.-H.; Zanni, M. T. *Physical Chemistry Chemical Physics* **2009**, *11*, 748 (cit. on pp. 58, 59).
67. Frisch, M.; Trucks, G.; Schlegel, H.; Scuseria, G.; Robb, M.; Cheeseman, J.; Scalmani, G.; Barone, V.; Petersson, G.; Nakatsuji, H, et al. Gaussian 16 Revision A. 03. 2016; Gaussian Inc (cit. on pp. 60, 101).
68. Woutersen, S.; Mu, Y.; Stock, G.; Hamm, P. *Proceedings of the National Academy of Sciences* **2001**, *98*, 11254–11258 (cit. on p. 64).
69. Weeks, J. D.; Chandler, D.; Andersen, H. C. *The Journal of Chemical Physics* **1971**, *54*, 5237–5247 (cit. on p. 76).
70. Kröger, M. *Journal of Non-Newtonian Fluid Mechanics* **2015**, *223*, 77–87 (cit. on p. 79).
71. Adams, D. J.; Rasaiah, J. C. *Faraday Discussions of the Chemical Society* **1977**, *64*, 22 (cit. on p. 80).

72. Martin, M. G.; Siepmann, J. I. *The Journal of Physical Chemistry B* **1998**, *102*, 2569–2577 (cit. on p. 90).
73. Case, D.; Babin, V; Berryman, J; Betz, R.; Cai, Q; Cerutti, D.; Cheatham Iii, T; Darden, T; Duke, R; Gohlke, H, et al. AMBER14 Package (cit. on p. 103).
74. Pronk, S.; Páll, S.; Schulz, R.; Larsson, P.; Bjelkmar, P.; Apostolov, R.; Shirts, M. R.; Smith, J. C.; Kasson, P. M.; Van Der Spoel, D., et al. *Bioinformatics* **2013**, *29*, 845–854 (cit. on p. 104).

# Appendices

# 1 Prepare Molecule for AMBER MD Simulation

## 1.1 Making a Molecule in Gaussian

Save molecule built in gview as 'filename.gjf'. This gives a file with atom types and positions/bondings. Make sure that the first section of code has the following lines:

```
%nprocshared=6
%mem=20GB
%RWF=filename.rwf
%NoSave
%chk=filename.chk
# b3lyp/6-311g* opt symmetry=None

comment line

0 1
C x y z
...
```

The first set of lines beginning with the '%' character are lines that contain different settings that pertain to how the job should be run on the computer. The first line is the number of processors used including multi-threading, i.e. a quad-core CPU with multi-threading could have a total of 8 for its value of `nprocshared`. The second line is the amount of memory the calculation can use. Sometimes when a segmentation fault error occurs, decreasing this value makes the calculation run more efficiently with regard to memory usage, and the error can be avoided. The third line is the name of the read-write-file which is a file Gaussian uses to store information over the course of the calculation. The `NoSave` line just says to not save the read-write-file once the calculation is complete. The following line sets the file name for the check-file which is a non-human-readable file with additional information that is not kept in the log-file.

The line preceded by the '#' character contains the level of theory and type of calculation that is to be run. The functional and the basis set are the first two terms, `opt` is the type of calculation, and any additional arguments can be set such as `symmetry`. Then there is a blank line, a comment line, another blank line, and then the charge and multiplicity followed by the atom types and coordinates.

The coordinates section should always be followed by a blank line, even when there are no other sections in the job-file.

Run the job using Gaussian 16<sup>67</sup> with the following command in a bash terminal:

```
g16 < filename.gjf > filename.log
```

<sup>67</sup>Frisch, M. et al. Gaussian 16 Revision A. 03. 2016; Gaussian Inc.

### 1.1.1 Frequency Calculation

Take the final coordinates from the optimization log-file and paste them over the old coordinates in an identical format to the geometry optimization job-file. Rename the `chk`, `rwf`, and `gjf` accordingly so that they are distinguished as being the part of the *frequency* calculation. Now, the line that is preceded by the '#' character has to be changed to run a frequency calculation.

```
#p b3lyp/6-311g* freq=(noraman,hpmodes) symmetry=none
```

where the `hpmodes` flag denotes high-precision modes. The `symmetry` flag is essential if transition-dipole-moments i.e. 'dipole derivatives' are going to be used in analysis at all.

### 1.1.2 Generate Electro-Static Potential from Optimized Geometry

From optimized geometry, generate an electrostatic potential (ESP) using Hartree-Fock 6-31g\* basis.

```
#p hf/6-31g* SCF=Tight Pop=MK IOp(6/33=2)
```

Next, charges are found using the ESP. Extract ESP info from log-file into a mol2 file format,

```
espgen -i filename_esp.log -o filename.esp
```

Convert the Gaussian output file into an antechamber file format.

```
antechamber -i filename_esp.log -fi gout -o filename.ac -fo ac
```

Create the RESP input files. The second command is for molecules have degenerate atoms with identical charges.

```
respgen -i filename.ac -o filename.respin1 -f resp1  
respgen -i filename.ac -o filename.respin2 -f resp2
```

At this point, the `respin1` and `respin2` files should be checked. The flags at the beginning should be fine. The atom list first gives the total charge and number of atoms. Then each atom is given a mass and degeneracy. `respin1` should have all unique charges, i.e. a value of 0. `respin2` connects degenerate atoms.

Finally, the charges are calculated:

```
resp -0 -i filename.respin1 -o filename.respout1 -e filename.esp -t filename.qout1  
resp -0 -i filename.respin2 -o filename.respout2 -e filename.esp -q filename.qout1  
-t filename.qout2
```

Use `antechamber` to combine the geometry and charges together into a mol2 file.

```
antechamber -i filename.ac -fi ac -o filename.mol2 -fo mol2 -c rc -cf filename.  
qout2
```

---

If an error occurs at this step it may be due to the fact that Amber14 and AmberTools15<sup>73</sup> are not being used. These instructions were originally formulated for Amber14. However, an error entitled 'weird valence error' may come up if using the Amber16 and AmberTools17<sup>7</sup>. If this happens, run the Gaussian ESP calculation but skip the rest of the commands previously listed and instead run these two commands to make the mol2 file:

```
antechamber -i filename_esp.log -fi gout -o filename.mol2 -fo mol2
antechamber -i filename.mol2 -fi mol2 -o filename.mol2 -fo mol2 -c rc -cf filename.
qout1
```

---

A name needs to be prescribed for the molecule which is generically named 'MOL'. For this example we will choose the name 'ABC'.

```
# on linux
sed -i -e "s/MOL/ABC/g" -e "s/ABCECULE/MOLECULE/g" filename.mol2
# on mac
sed -e "s/MOL/ABC/g" -e "s/ABCECULE/MOLECULE/g" filename.mol2 > filename.mol2
```

The command parmchk2 finds appropriate force fields for the molecules that are not standard.

```
parmchk2 -i filename.mol2 -f mol2 -o filename.frcmod
```

Then load the molecule into tleap with the following commands:

```
tleap
> source leaprc.ff14SB
> source leaprc.gaff
> ABC = loadmol2 filename.mol2
> loadamberparams filename.frcmod
> saveoff ABC filename.lib
> loadoff solvents.lib
```

Put all of these commands into a setup-file and then just run 'source setup-file' in tleap. The '>' characters signify a command entered into the tleap terminal and should be omitted from setup-file. ABC is the residue name as before. The filename's are the files that were created in previous steps. The last line is only needed in Amber16 and AmberTools17 since some solvents are not loaded by default.

If duplicates of the molecule are desired we can make another file called 'duplicate-file' and source it in tleap with the following contents or we can just enter into tleap:

---

<sup>73</sup>Case, D. et al. AMBER14 Package.

<sup>7</sup>Case, D. et al. AMBER16 Package.

```
tLeap
> test = combine {ABC ABC}
> translate test.2 {x y z}
```

The number 2 after test indicates that the second ABC residue in the unit 'test' will be translated by x, y, and z. So this process can be extrapolated for more ABC residues.

Next, show the box size required to fit just the residues in 'test'. In this example we will add use the chloroform solvent from Amber,

```
> solvateShell test CHCL3BOX 0.1
```

And then add solvent. The solvateBox command adds  $x \text{ \AA}$  of solvent between the box  $x$  edge and the  $x$  edge given from the solvateShell command. Boxes should be roughly cubic. GPU simulations require  $\geq 7500$  atoms.

```
> solvateBox test CHCL3BOX {x y z}
```

If this command gives an error about not recognizing an atom type in the solvent, run:

```
> CHCL3BOX = loadamberparams frcmod.chcl3
```

And then, finally run:

```
saveAmberParm test test.prmtop test.inpcrd
```

The prmtop and inpcrd files are the parameter and starting coordinate files, respectively, that we will use in our Amber MD simulations.

For non-standard solvent, add in solvent with GROMACS<sup>74</sup>. The following command creates a box with 7 nm sides and attempts to fit 1500 molecules from 'solvent\_name.pdb'.

```
antechamber -i filename.mol2 -fi mol2 -o filename.pdb -fo pdb
```

```
gmx insert-molecules -ci solvent_name.pdb -nmol 1500 -box 7 7 7 -o thf-box.pdb
```

The .gro file might be able to be written as a .pdb file but if not, use VMD to convert the .gro to a .pdb file. Then, make a new set of AMBER prmtop and inpcrd files by entering the following with tLeap.

```
load leaprc.gaff
source leaprc.gaff
BUT = loadmol2 BUT.mol2 # load molecule parameters
loadamberparams BUT.frcmod
saveoff BUT BUT.lib
x = loadpdb butane-box.pdb # load molecule coordinates with molecule name as above
setBox x "vdw"
saveAmberParm x test.prmtop test.inpcrd
```

<sup>74</sup>Pronk, S. et al. *Bioinformatics* 2013, 29, 845–854.



Then these AMBER files can be used to equilibrate the solvent box. Once we have an equilibrated solvent box, we can use GROMACS to solvate our solute in that new solvent.

```
gmx insert-molecules -ci PDI_vacuum_no_box.pdb -box 5.85 5.85 5.85 -nmol 1 -o
PDI_vacuum_box.pdb
gmx editconf -f PDI_vacuum_box.pdb -center 2.93 2.93 2.93 -o
PDI_vacuum_box_centered.pdb
gmx solvate -cp PDI_vacuum_box_centered.pdb -cs THF.pdb -o
PDI_solvated_box_centered.pdb
```

where 5.85 is the length in nanometers of the full solvent box sides. 2.93 is half that length, so as to place the PDI's in the center of the box.

THF.pdb is the equilibrated solvent box (only THF) with dimensions 5.85×5.85×5.85 nanometers.

---

### 1.1.3 Restraints

When adding a restraint into an AMBER simulation like a harmonic spring between two atoms. The atom numbers in the Amber restraint file are the VMD *index* value +1.

## 2 Code

### 2.1 Model System IS-SPA Histogram Measurement Code: gr2d.p2.py

```
# python3
# Compute Radial Distribution Function as a Function of r, cos(theta), and phi [
  Spread into 3D ] around an LJ sphere pair

import numpy as np
import sys
import os
import MDAnalysis
from math import *

def cosPhi_2_phi(cosPhi):
    if cosPhi > 1:
        return 0
    elif cosPhi < -1:
        return np.pi
    else:
        return np.arccos(cosPhi)

def wrap_phi(phi):
    if phi >= pi23:
        return phi-pi23, 1;
    elif pi3 < phi < pi23:
        return pi23-phi, -1;
    else:
        return phi, 1;

def compute_pbc_dr(r1,r2,box,hbox):
```

```

dr = r1 - r2 # dr points from r2 to r1
if dr < -hbox:
    dr += box
elif dr > hbox:
    dr -= box
return dr;

def t_dot_rcl(dot, sign):
    if dot < 0:
        return sign*(-1);
    else:
        return sign;

## Read configuration file and populate global variables
def parse_config_file(cfgFile):
    global topFile, trajFile, outFile, histDistMin, histDistMax, binDistSize,
    histThetaMin, histThetaMax, binThetaSize, T, soluteResname, solventResname, d,
    histPhiMin, histPhiMax, binPhiSize, nAtomTypes
    trajFile = []
    f = open(cfgFile)
    for line in f:
        # first remove comments
        if '#' in line:
            line, comment = line.split('#',1)
        if '=' in line:
            option, value = line.split('=',1)
            option = option.strip()
            value = value.strip()
            print("Option:", option, " Value:", value)
            # check value
            if option.lower()=='topfile':
                topFile = value
            elif option.lower()=='trajfile':
                trajFile.append(value)
            elif option.lower()=='outfile':
                outFile = value
            elif option.lower()=='hist_dist_min':
                histDistMin = float(value)
            elif option.lower()=='hist_dist_max':
                histDistMax = float(value)
            elif option.lower()=='bin_dist_size':
                binDistSize = float(value)
            elif option.lower()=='hist_theta_min':
                histThetaMin = float(value)
            elif option.lower()=='hist_theta_max':
                histThetaMax = float(value)
            elif option.lower()=='bin_theta_size':
                binThetaSize = float(value)
            elif option.lower()=='hist_phi_min':
                histPhiMin = float(value)
            elif option.lower()=='hist_phi_max':
                histPhiMax = float(value)
            elif option.lower()=='bin_phi_size':
                binPhiSize = float(value)
            elif option.lower()=='temperature':
                T = float(value)
            elif option.lower()=='solute_resname':
                soluteResname = value
            elif option.lower()=='solvent_resname':
                solventResname = value
            elif option.lower()=='offset':
                d = float(value)
            elif option.lower()=='number_solute_atoms':

```

```

        nAtomTypes = int(value)
    else :
        print("Option:", option, " is not recognized")

# set some extra global variables
global kT, histDistMin2, histDistMax2, nDistBins, nThetaBins, nPhiBins, pi23, pi3

# Boltzmann Constant in kcal/mol.K
k_B = 0.0019872041
kT = k_B * T

# Distances [in Angstroms]
histDistMin2= histDistMin*histDistMin
histDistMax2= histDistMax*histDistMax

# Histogram bins
nDistBins = int((histDistMax - histDistMin)/binDistSize)

# Cosine Theta Histogram bins
nThetaBins = int((histThetaMax - histThetaMin)/binThetaSize)
# Phi Histogram bins
nPhiBins = int((histPhiMax - histPhiMin)/binPhiSize)

# global constants
pi23 = 2*pi/3. # FIXME: this should be incorporated into the config file somehow.
Like a radian of symmetry and half of that value.
pi3 = pi/3.

f.close()

## Read prmtop file and populate global variables
def parse_prmtop_bonded(topFile):
    global bond_fc, bond_equil_values, angle_fc, angle_equil_values, dihedral_fc,
    dihedral_period, dihedral_phase, nbonh, nbona, ntheta, ntheth, nphia, nphih, bondsh, bondsa,
    anglesh, anglesa, dihedralsh, dihedralrsa, n_atoms, n_types, atom_names, atom_type_index,
    nb_parm_index, lj_a_coeff, lj_b_coeff

    param = open(topFile, 'r')
    pointers = np.zeros(31, dtype=int)
    lines = param.readlines()
    for i in range(len(lines)):
        if lines[i][0:14] == '%FLAG POINTERS':
            for j in range(4):
                temp = lines[i+2+j].split()
                for k in range(len(temp)):
                    pointers[j*10+k] = int(temp[k])
            n_atoms = pointers[0]
            n_types = pointers[1]
            nbonh = pointers[2]
            nbona = pointers[12]
            ntheth = pointers[4]
            ntheta = pointers[13]
            nphih = pointers[6]
            nphia = pointers[14]
            numbnd = pointers[15]
            numang = pointers[16]
            numtra = pointers[17]
            n_type_lines = int(ceil(n_atoms/10.))
            n_name_lines = int(ceil(n_atoms/20.))
            n_nb_parm_lines = int(ceil(n_types*n_types/10.))
            n_lj_param_lines = int(ceil((n_types*(n_types+1)/2)/5.))
            n_bond_lines = int(ceil(numbnd/5.))
            n_angle_lines = int(ceil(numang/5.))
            n_dihedral_lines = int(ceil(numtra/5.))

```

```

n_bondsh_lines = int(ceil(nbonh*3/10.))
n_bondsa_lines = int(ceil(nbona*3/10.))
n_anglesh_lines = int(ceil(ntheth*4/10.))
n_anglesa_lines = int(ceil(ntheta*4/10.))
n_dihedralsh_lines = int(ceil(nphih*5/10.))
n_dihedralsa_lines = int(ceil(nphia*5/10.))
bond_fc = np.zeros(numbnd, dtype=float)
bond_equil_values = np.zeros(numbnd, dtype=float)
angle_fc = np.zeros(numang, dtype=float)
angle_equil_values = np.zeros(numang, dtype=float)
dihedral_fc = np.zeros(numtra, dtype=float)
dihedral_period = np.zeros(numtra, dtype=float)
dihedral_phase = np.zeros(numtra, dtype=float)
SCE_factor = np.zeros(numtra, dtype=float)
SCNB_factor = np.zeros(numtra, dtype=float)
bondsh_linear = np.zeros(3*nbonh, dtype=int)
bondsa_linear = np.zeros(3*nbona, dtype=int)
bondsh = np.zeros((nbonh, 3), dtype=int)
bondsa = np.zeros((nbona, 3), dtype=int)
anglesh_linear = np.zeros(4*ntheth, dtype=int)
anglesa_linear = np.zeros(4*ntheta, dtype=int)
anglesh = np.zeros((ntheth, 4), dtype=int)
anglesa = np.zeros((ntheta, 4), dtype=int)
dihedralsh_linear = np.zeros(5*nphih, dtype=int)
dihedralsa_linear = np.zeros(5*nphia, dtype=int)
dihedralsh = np.zeros((nphih, 5), dtype=int)
dihedralsa = np.zeros((nphia, 5), dtype=int)
atom_names = []
atom_type_index = np.zeros((n_atoms), dtype=int)
nb_parm_index = np.zeros(n_types*n_types, dtype=int)
lj_a_coeff = np.zeros((n_types*(n_types+1))/2, dtype=float)
lj_b_coeff = np.zeros((n_types*(n_types+1))/2, dtype=float)

if lines[i][0:25] == '%FLAG BOND_FORCE_CONSTANT':
    for j in range(n_bond_lines):
        temp = lines[i+2+j].split()
        for k in range(len(temp)):
            bond_fc[j*5+k] = float(temp[k])
if lines[i][0:22] == '%FLAG BOND_EQUIL_VALUE':
    for j in range(n_bond_lines):
        temp = lines[i+2+j].split()
        for k in range(len(temp)):
            bond_equil_values[j*5+k] = float(temp[k])
if lines[i][0:26] == '%FLAG ANGLE_FORCE_CONSTANT':
    for j in range(n_angle_lines):
        temp = lines[i+2+j].split()
        for k in range(len(temp)):
            angle_fc[j*5+k] = float(temp[k])
if lines[i][0:23] == '%FLAG ANGLE_EQUIL_VALUE':
    for j in range(n_angle_lines):
        temp = lines[i+2+j].split()
        for k in range(len(temp)):
            angle_equil_values[j*5+k] = float(temp[k])
if lines[i][0:29] == '%FLAG DIHEDRAL_FORCE_CONSTANT':
    for j in range(n_dihedral_lines):
        temp = lines[i+2+j].split()
        for k in range(len(temp)):
            dihedral_fc[j*5+k] = float(temp[k])
if lines[i][0:26] == '%FLAG DIHEDRAL_PERIODICITY':
    for j in range(n_dihedral_lines):
        temp = lines[i+2+j].split()
        for k in range(len(temp)):
            dihedral_period[j*5+k] = float(temp[k])
if lines[i][0:20] == '%FLAG DIHEDRAL_PHASE':
    for j in range(n_dihedral_lines):

```

```

        temp = lines[i+2+j].split()
        for k in range(len(temp)):
            dihedral_phase[j*5+k] = float(temp[k])
if lines[i][0:23] == '%FLAG SCEE_SCALE_FACTOR':
    for j in range(n_dihedral_lines):
        temp = lines[i+2+j].split()
        for k in range(len(temp)):
            SCEE_factor[j*5+k] = float(temp[k])
if lines[i][0:23] == '%FLAG SCNB_SCALE_FACTOR':
    for j in range(n_dihedral_lines):
        temp = lines[i+2+j].split()
        for k in range(len(temp)):
            SCNB_factor[j*5+k] = float(temp[k])
if lines[i][0:24] == '%FLAG BONDS_INC_HYDROGEN':
    for j in range(n_bondsh_lines):
        temp = lines[i+2+j].split()
        for k in range(len(temp)):
            bondsh_linear[j*10+k] = int(temp[k])
    for j in range(nbonh):
        bondsh[j][0] = bondsh_linear[j*3]
        bondsh[j][1] = bondsh_linear[j*3+1]
        bondsh[j][2] = bondsh_linear[j*3+2]
if lines[i][0:28] == '%FLAG BONDS_WITHOUT_HYDROGEN':
    for j in range(n_bondsa_lines):
        temp = lines[i+2+j].split()
        for k in range(len(temp)):
            bondsa_linear[j*10+k] = int(temp[k])
    for j in range(nbona):
        bondsa[j][0] = bondsa_linear[j*3]
        bondsa[j][1] = bondsa_linear[j*3+1]
        bondsa[j][2] = bondsa_linear[j*3+2]
if lines[i][0:25] == '%FLAG ANGLES_INC_HYDROGEN':
    for j in range(n_anglesh_lines):
        temp = lines[i+2+j].split()
        for k in range(len(temp)):
            anglesh_linear[j*10+k] = int(temp[k])
    for j in range(ntheth):
        anglesh[j][0] = anglesh_linear[j*4]
        anglesh[j][1] = anglesh_linear[j*4+1]
        anglesh[j][2] = anglesh_linear[j*4+2]
        anglesh[j][3] = anglesh_linear[j*4+3]
if lines[i][0:29] == '%FLAG ANGLES_WITHOUT_HYDROGEN':
    for j in range(n_anglesa_lines):
        temp = lines[i+2+j].split()
        for k in range(len(temp)):
            anglesa_linear[j*10+k] = int(temp[k])
    for j in range(ntheta):
        anglesa[j][0] = anglesa_linear[j*4]
        anglesa[j][1] = anglesa_linear[j*4+1]
        anglesa[j][2] = anglesa_linear[j*4+2]
        anglesa[j][3] = anglesa_linear[j*4+3]
if lines[i][0:28] == '%FLAG DIHEDRALS_INC_HYDROGEN':
    for j in range(n_dihedralsh_lines):
        temp = lines[i+2+j].split()
        for k in range(len(temp)):
            dihedralsh_linear[j*10+k] = int(temp[k])
    for j in range(nphih):
        dihedralsh[j][0] = dihedralsh_linear[j*5]
        dihedralsh[j][1] = dihedralsh_linear[j*5+1]
        dihedralsh[j][2] = dihedralsh_linear[j*5+2]
        dihedralsh[j][3] = dihedralsh_linear[j*5+3]
        dihedralsh[j][4] = dihedralsh_linear[j*5+4]
if lines[i][0:32] == '%FLAG DIHEDRALS_WITHOUT_HYDROGEN':
    for j in range(n_dihedralsa_lines):
        temp = lines[i+2+j].split()

```

```

        for k in range(len(temp)):
            dihedral_linear[j*10+k] = int(temp[k])
    for j in range(nphia):
        dihedral_linear[j][0] = dihedral_linear[j*5]
        dihedral_linear[j][1] = dihedral_linear[j*5+1]
        dihedral_linear[j][2] = dihedral_linear[j*5+2]
        dihedral_linear[j][3] = dihedral_linear[j*5+3]
        dihedral_linear[j][4] = dihedral_linear[j*5+4]
    if lines[i][0:15] == '%FLAG ATOM_NAME':
        for j in range(n_name_lines):
            temp = lines[i+2+j].split()
            for k in range(len(temp)):
                atom_names.append(temp[k])
    if lines[i][0:21] == '%FLAG ATOM_TYPE_INDEX':
        for j in range(n_type_lines):
            temp = lines[i+2+j].split()
            for k in range(len(temp)):
                atom_type_index[j*10+k] = float(temp[k])
    if lines[i][0:26] == '%FLAG NONBONDED_PARM_INDEX':
        for j in range(n_nb_parm_lines):
            temp = lines[i+2+j].split()
            for k in range(len(temp)):
                nb_parm_index[j*10+k] = float(temp[k])
    if lines[i][0:25] == '%FLAG LENNARD_JONES_ACOEF':
        for j in range(n_lj_param_lines):
            temp = lines[i+2+j].split()
            for k in range(len(temp)):
                lj_a_coeff[j*5+k] = float(temp[k])
    if lines[i][0:25] == '%FLAG LENNARD_JONES_BCOEF':
        for j in range(n_lj_param_lines):
            temp = lines[i+2+j].split()
            for k in range(len(temp)):
                lj_b_coeff[j*5+k] = float(temp[k])

# initialize arrays for 3D g(r,cos(theta),phi)
def initialize_arrays():
    # NOTE
    # g(r) array has both the g(r) values and the counts for each atomtype.
    # Gc[atomtypes, distbin, thetabin, phibin]
    Gc = np.zeros((nAtomTypes, nDistBins, nThetaBins, nPhiBins), dtype=float)
    Gr = np.zeros((nAtomTypes, nDistBins, nThetaBins, nPhiBins), dtype=float)
    # NOTE
    # Force array has the force, its square, the std.dev. of the force, and its square
    # for each atomtype.
    # FrLJ[atomtypes, <f.r>/<f.s>/<f.t>, distbin, thetabin, phibin]
    FrLJ = np.zeros((nAtomTypes, 3, nDistBins, nThetaBins, nPhiBins), dtype=float)
    # FrC[atomtypes, <f.r>/<f.s>/<f.t>, distbin, thetabin, phibin]
    FrC = np.zeros((nAtomTypes, 3, nDistBins, nThetaBins, nPhiBins), dtype=float)
    return Gc, Gr, FrLJ, FrC;

# loop through trajectory
def iterate(Gc, FrLJ, FrC):
    u = MDAnalysis.Universe(topFile, trajFile[0]) # initiate MDAnalysis Universe.
    solvSel = u.select_atoms('resname ' + soluteResname)
    solvSel = u.select_atoms('resname ' + solventResname)
    nSolv = len(solvSel) # number of solvent atoms
    cosPhi2Phi = np.vectorize(cosPhi_2_phi) # cosPhi -> phi conditions
    wrapPhi = np.vectorize(wrap_phi) # phi -> wrapped phi conditions
    wrapPbc = np.vectorize(compute_pbc_dr) # wrap vector difference in pbc box
    tDotRcl = np.vectorize(t_dot_rcl) # change sign based on t vector orientation
    for igo in range(len(trajFile)):
        u.load_new(trajFile[igo])

```

```

print("Now analyzing trajectory file: ", trajFile[igo])
for ts in u.trajectory: # Loop over all time steps in the trajectory.
    # Progress Bar
    sys.stdout.write("Progress: {0:.2f}% Complete\r".format((ts.frame+len(u.
trajectory)*(igo))/(len(u.trajectory)*len(trajFile))*100))
    sys.stdout.flush()

    box = u.dimensions[:3]## define box and half box here so we save division
calculation for every distance pair when we calculate 'dist' below.
    hbox = u.dimensions[:3]/2

    # Compute all pairwise distances
    if nAtomTypes == 2:
        ljDr = wrapPbc(soluSel.atoms[0].position, soluSel.atoms[1].position,
box, hbox)# Calculate the vector (ljDr) between the two LJ particles.
        for a in soluSel.atoms:
            # Calculate r,cosTh,phi bin
            pCH = wrapPbc(solvSel.atoms[np.arange(0,nSolv,5)].positions, solvSel.
atoms[np.arange(1,nSolv,5)].positions, box, hbox)
            np.divide( pCH, np.sqrt(np.einsum('ij,ij->i',pCH,pCH))[:,None], out=
pCH) # normalize pCH
            rSolv = solvSel.atoms[np.arange(1,nSolv,5)].positions - d*pCH
            rSolv = wrapPbc(a.position, rSolv, box, hbox)
            pCCL = wrapPbc(solvSel.atoms[np.arange(2,nSolv,5)].positions, solvSel.
atoms[np.arange(1,nSolv,5)].positions, box, hbox)

            if nAtomTypes == 2:
                if a.index == 0:
                    ip = np.where( np.dot(rSolv,ljDr)>0 ) # far side from solute
                elif a.index == 1:
                    ip = np.where( np.dot(rSolv,ljDr)<0 ) # far side from solute
                rSolv = np.delete(rSolv,ip,axis=0)
                pCH = np.delete(pCH,ip,axis=0)
                pCCL = np.delete(pCCL,ip,axis=0)

            ir = np.where( np.einsum('ij,ij->i',rSolv,rSolv) >= histDistMax2 )
            rSolv = np.delete(rSolv,ir,axis=0)
            pCH = np.delete(pCH,ir,axis=0)
            pCCL = np.delete(pCCL,ir,axis=0)

            rSolvDist = np.sqrt(np.einsum('ij,ij->i',rSolv,rSolv))
            cosTh = np.divide( np.einsum('ij,ij->i',pCH,rSolv), rSolvDist) # +1
when pCH points toward solute

            nLJCH = np.cross(pCH, rSolv)
            nHCCl1 = np.cross(pCH, pCCL)
            cosPhi = np.divide( np.einsum('ij,ij->i',nLJCH,nHCCl1), np.sqrt(np.
einsum('ij,ij->i',nLJCH,nLJCH)*np.einsum('ij,ij->i',nHCCl1,nHCCl1)) )
            cosPhi = cosPhi2Phi( cosPhi ) # apply cosPhi_2_phi to cosPhi, cosPhi
is now converted into phi
            phi,sign = wrapPhi( cosPhi ) # apply wrap_phi to phi

            distBin = np.ndarray.astype( rSolvDist/binDistSize, np.int)
            thetaBin = np.ndarray.astype( (cosTh-histThetaMin)/binThetaSize, np.
int)

            phiBin = np.ndarray.astype( (phi-histPhiMin)/binPhiSize, np.int)
            distBin[distBin == nDistBins] = -1
            thetaBin[thetaBin == nThetaBins] = -1
            phiBin[phiBin == nPhiBins] = -1

            # compute lj and coulomb force atom-by-atom
            solvAtomInd = solvSel.atoms[np.arange(0,nSolv,1)].indices
            solvAtomPos = solvSel.atoms[np.arange(0,nSolv,1)].positions
            solvAtomChg = solvSel.atoms[np.arange(0,nSolv,1)].charges

```

```

# Delete all atoms associated with residues that have been deleted
previously above before calculating force to save time
if nAtomTypes == 2:
    ipa = np.append(np.append(np.append(np.append( ip[0]*5, ip[0]*5+1)
, ip[0]*5+2), ip[0]*5+3), ip[0]*5+4) # list of atoms to remove based on "residue"
dot product removal list
    solvAtomInd = np.delete(solvAtomInd,ipa,axis=0) # remove atoms
that belong to previously deleted residues: dot product
    solvAtomPos = np.delete(solvAtomPos,ipa,axis=0) # remove atoms
that belong to previously deleted residues: dot product
    solvAtomChg = np.delete(solvAtomChg,ipa,axis=0) # remove atoms
that belong to previously deleted residues: dot product
    ira = np.append(np.append(np.append(np.append( ir[0]*5, ir[0]*5+1), ir
[0]*5+2), ir[0]*5+3), ir[0]*5+4) # list of atoms to remove based on "residue"
distance removal list
    solvAtomInd = np.delete(solvAtomInd,ira,axis=0) # remove atoms that
belong to previously deleted residues: distance
    solvAtomChg = np.delete(solvAtomChg,ira,axis=0) # remove atoms that
belong to previously deleted residues: distance
    solvAtomPos = np.delete(solvAtomPos,ira,axis=0) # remove atoms that
belong to previously deleted residues: distance
    index = n_types*(atom_type_index[a.index]-1) + atom_type_index[
solvAtomInd]-1
    nbIndex = nb_parm_index[index]-1
    rSolvAtom = wrapPbc(a.position, solvAtomPos, box, hbox)
    rSolvAtomDist2 = np.einsum('ij,ij->i',rSolvAtom,rSolvAtom)
    r6 = np.power(rSolvAtomDist2, -3)
    # LJ
    fSolvAtomLJ = np.einsum('i,ij->ij',(r6 * (12*r6*lj_a_coeff[nbIndex] -
6*lj_b_coeff[nbIndex])) / rSolvAtomDist2), rSolvAtom) # force vectors
    fSolvLJ = fSolvAtomLJ[np.arange(0,len(fSolvAtomLJ),5)] + fSolvAtomLJ[
np.arange(1,len(fSolvAtomLJ),5)] + fSolvAtomLJ[np.arange(2,len(fSolvAtomLJ),5)] +
fSolvAtomLJ[np.arange(3,len(fSolvAtomLJ),5)] + fSolvAtomLJ[np.arange(4,len(
fSolvAtomLJ),5)] # summed force vector from residue atoms
    # Coulomb
    fSolvAtomC = 332.05595 * a.charge * solvAtomChg[:,None] * rSolvAtom *
np.sqrt(r6)[:,None]
    fSolvC = fSolvAtomC[np.arange(0,len(fSolvAtomC),5)] + fSolvAtomC[np.
arange(1,len(fSolvAtomC),5)] + fSolvAtomC[np.arange(2,len(fSolvAtomC),5)] +
fSolvAtomC[np.arange(3,len(fSolvAtomC),5)] + fSolvAtomC[np.arange(4,len(fSolvAtomC)
,5)] # summed force vector from residue atoms
    tSolv = np.cross(rSolv,pCH) # t vector
    np.divide(tSolv, np.sqrt(np.einsum('ij,ij->i',tSolv,tSolv))[:,None],
out=tSolv) # normalize t vector
    tDot = np.einsum('ij,ij->i',tSolv,pCC1)
    sign = tDotRcl(tDot,sign)
    sSolv = np.cross(tSolv,rSolv) # s vector
    np.divide(sSolv, np.sqrt(np.einsum('ij,ij->i',sSolv,sSolv))[:,None],
out=sSolv) # normalize s vector
    fLJr = np.einsum('ij,ij->i',fSolvLJ,rSolv)/rSolvDist # force along r
    fLJs = np.einsum('ij,ij->i',fSolvLJ,sSolv)
    fLJt = np.einsum('ij,ij->i',fSolvLJ,tSolv)
    fCr = np.einsum('ij,ij->i',fSolvC,rSolv)/rSolvDist # force along r
    fCs = np.einsum('ij,ij->i',fSolvC,sSolv)
    fCt = np.einsum('ij,ij->i',fSolvC,tSolv)

    np.add.at(Gc[a.index], tuple(np.stack((distBin,thetaBin,phiBin))), 1)
    np.add.at(FrLJ[a.index][0], tuple(np.stack((distBin,thetaBin,phiBin)))
, fLJr)
    np.add.at(FrLJ[a.index][1], tuple(np.stack((distBin,thetaBin,phiBin)))
, fLJs)
    np.add.at(FrLJ[a.index][2], tuple(np.stack((distBin,thetaBin,phiBin)))
, fLJt)
    np.add.at(FrC[a.index][0], tuple(np.stack((distBin,thetaBin,phiBin))),
fCr)

```



```

fCs)          np.add.at(FrC[a.index][1], tuple(np.stack((distBin, thetaBin, phiBin))),
fCt)          np.add.at(FrC[a.index][2], tuple(np.stack((distBin, thetaBin, phiBin))),

def average_Fr(Gc, FrLJ, FrC):
# Average LJ radial force for each distance and cos(theta) bin
for a in range(nAtomTypes):
    np.divide(FrLJ[a][0], Gc[a], out=FrLJ[a][0], where=Gc[a][:,:,:]!=0)
    np.divide(FrLJ[a][1], Gc[a], out=FrLJ[a][1], where=Gc[a][:,:,:]!=0)
    np.divide(FrLJ[a][2], Gc[a], out=FrLJ[a][2], where=Gc[a][:,:,:]!=0)
    np.divide(FrC[a][0], Gc[a], out=FrC[a][0], where=Gc[a][:,:,:]!=0)
    np.divide(FrC[a][1], Gc[a], out=FrC[a][1], where=Gc[a][:,:,:]!=0)
    np.divide(FrC[a][2], Gc[a], out=FrC[a][2], where=Gc[a][:,:,:]!=0)

def volume_correct(Gc, Gr):
## Volume Correct
for a in range(nAtomTypes):
    for i in range(nDistBins):
        for j in range(nThetaBins):
            for k in range(nPhiBins):
                Gr[a, i, j, k] = Gc[a,i,j,k] / (4*pi*((i+0.5)*binDistSize +
histDistMin)**2)

def normalize_Gr(Gr):
## Normalize
## have to normalize after volume correction because the 'bulk' g(r) value changes
after volume correction.
norm_points = 10
for a in range(nAtomTypes):
    # normalize by the last 'norm_points' distance points
    g_norm = 0.
    for k in range(nPhiBins):
        for j in range(nThetaBins):
            for i in range(norm_points):
                g_norm += Gr[a, -(i+1), j, k]

    g_norm /= float(norm_points*nThetaBins*nPhiBins)

    for k in range(nPhiBins):
        for j in range(nThetaBins):
            for i in range(nDistBins):
                Gr[a, i, j, k] /= g_norm

def write_out_1(outFile, Gc, Gr, FrLJ, FrC):
## Open Output File
out = open(outFile, 'w')

out.write("## 1: Distance Bin\n")
out.write("## 2: Cos(theta) Bin\n")
out.write("## 3: Phi/3 Bin\n")
out.write("## 4: g(r)\n")
out.write("## 5: <fLJ . r>\n")
out.write("## 6: <fLJ . s>\n")
out.write("## 7: <fLJ . t>\n")
out.write("## 8: <fC . r>\n")
out.write("## 9: <fC . s>\n")
out.write("## 10: <fC . t>\n")
out.write("## 11: g(r) Counts\n")
for i in range(nDistBins):
    for j in range(nThetaBins):

```

```

        for k in range(nPhiBins):
            out.write("%10.5f %10.5f %10.5f %10.5f %10.5f %10.5f %10.5f %10.5f
%10.5f %10.5f %10.5f\n" %((i+0.5)*binDistSize+histDistMin, (j+0.5)*binThetaSize+
histThetaMin, (k+0.5)*binPhiSize+histPhiMin, Gr[0,i,j,k], FrLJ[0,0,i,j,k], FrLJ
[0,1,i,j,k], FrLJ[0,2,i,j,k], FrC[0,0,i,j,k], FrC[0,1,i,j,k], FrC[0,2,i,j,k], Gc[0,
i,j,k]))

    ## Close Output File
    out.close

def write_out_2(outFile, Gc, Gr, FrLJ, FrC):
    ## Open Output File
    out = open(outFile, 'w')

    out.write("## 1: Distance Bin\n")
    out.write("## 2: Cos(theta) Bin\n")
    out.write("## 3: Phi/3 Bin\n")
    out.write("## 4: g(r) +\n")
    out.write("## 5: g(r) -\n")
    out.write("## 6: <fLJ . r> +\n")
    out.write("## 7: <fLJ . s> +\n")
    out.write("## 8: <fLJ . t> +\n")
    out.write("## 9: <fLJ . r> -\n")
    out.write("## 10: <fLJ . s> -\n")
    out.write("## 11: <fLJ . t> -\n")
    out.write("## 12: <fC . r> +\n")
    out.write("## 13: <fC . s> +\n")
    out.write("## 14: <fC . t> +\n")
    out.write("## 15: <fC . r> -\n")
    out.write("## 16: <fC . s> -\n")
    out.write("## 17: <fC . t> -\n")
    out.write("## 18: g(r) Counts +\n")
    out.write("## 19: g(r) Counts -\n")
    for i in range(nDistBins):
        for j in range(nThetaBins):
            for k in range(nPhiBins):
                out.write("%10.5f %10.5f %10.5f %10.5f %10.5f %10.5f %10.5f %10.5f
%10.5f %10.5f %10.5f %10.5f %10.5f %10.5f %10.5f %10.5f %10.5f\n" %((
i+0.5)*binDistSize+histDistMin, (j+0.5)*binThetaSize+histThetaMin, (k+0.5)*
binPhiSize+histPhiMin, Gr[0,i,j,k], Gr[1,i,j,k], FrLJ[0,0,i,j,k], FrLJ[0,1,i,j,k],
FrLJ[0,2,i,j,k], FrLJ[1,0,i,j,k], FrLJ[1,1,i,j,k], FrLJ[1,2,i,j,k], FrC[0,0,i,j,k],
FrC[0,1,i,j,k], FrC[0,2,i,j,k], FrC[1,0,i,j,k], FrC[1,1,i,j,k], FrC[1,2,i,j,k], Gc
[0,i,j,k], Gc[1,i,j,k]))

    ## Close Output File
    out.close

# main program
def mainLJ():
    # read in command line argument (cfg file)
    cfgFile = sys.argv[1]

    print('Reading input and initializing')
    # read cfg file
    parse_config_file(cfgFile)

    # parse the prmtop file
    parse_prmtop_bonded(topFile)

    #####
    # initialize 2D arrays

    # initialize with total dist, theta, and phi bins

```

```

Gc,Gr,FrLJ,FrC = initialize_arrays()

# loop through trajectory and calculate g(r,cos[theta]), force(r,cos[theta]),
boltzmann(r,cos[theta])
print('Looping through trajectory time steps...')
iterate(Gc, FrLJ, FrC)

# average the force and boltzmann by the g(r,cos[theta])
print('Volume correcting...')
average_Fr(Gc, FrLJ, FrC)

# volume correct g(r,cos[theta])
volume_correct(Gc, Gr)

# normalize g(r,cos[theta])
normalize_Gr(Gr)

# write 2D output file: Gc, frc, boltz, integrated_force
print('Write 2D output file')
if nAtomTypes == 1:
    write_out_1(outFile, Gc, Gr, FrLJ, FrC)
elif nAtomTypes == 2:
    write_out_2(outFile, Gc, Gr, FrLJ, FrC)

print('All Done!')

# Run Main program code.
mainLJ()

```

## 2.2 Chemical System IS-SPA 3D Histogram Measurement Code: gr3d.p2.py

```

# python3

import numpy as np
import MDAnalysis
import time
import math
import sys

# FILE VARIABLES
configFile = sys.argv[1]

psf = None
outname = None
coordDcd = None
forceDcd = None
param = None
coord = None
force = None
temperature = None
dims = None
hdims = None

debug = False
junkCounter = 0 # counter used for debugging

# DEFAULT GLOBAL VARIABLES
rMin = 0
rMax = 25
binSize = 0.1
binCount = 0
elecPermittivity = 8.854e-12 # electrical permittivity of vacuum C^2/Jm

```

```

boltzmann = 1.9872041e-3          # boltzmann constant in kcal/(K mol)
rho = 0
epsilon = []                      # list of epsilon values for all non-solvent atoms
lj_rMin = []                      # list of rMin values for all non-solvent atoms
d = 0                             # offset from central carbon to center of volume exclusion

# Set debug mode from config file
def setDebug(cF):
    global debug
    txt = open(cF, 'r')
    line = txt.readline()
    while line != "END CONFIG\n":
        if line == "DEBUG MODE: ON\n":
            debug = True
        line = txt.readline()

# Get name of PSF file from config file
def getPsf(cF):
    global psf
    print('\n\t***DCD Analysis***')
    if debug:
        print('\t\tDebug Mode ON')
    else:
        print('\t\tDebug Mode OFF')
    txt = open(cF, 'r')
    while psf is None:
        line = txt.readline()
        if line == 'PSF FILE:\n':
            psf = txt.readline()[:-1]
            if debug:
                print('PSF File: {}'.format(psf))
        elif line == 'END CONFIG FILE\n':
            print('No PSF file found in config.')
            break

# Get name of PSF file from config file
def getOut(cF):
    global outname
    txt = open(cF, 'r')
    while outname is None:
        line = txt.readline()
        if line == 'OUTPUT FILE NAME:\n':
            outname = txt.readline()[:-1]
            if debug:
                print('OUTPUT File: {}'.format(outname))
        elif line == 'END CONFIG FILE\n':
            print('No OUTPUT file found in config.')
            break

# Get name of Coordinate DCD files from config file
def getCoordDCDs(cF):
    global coordDcd
    coordDcd = []
    txt = open(cF, 'r')
    while len(coordDcd) == 0:
        line = txt.readline()
        if line == 'COORD DCD FILES:\n':
            line = txt.readline()
            while line != '\n':
                if line == 'END CONFIG\n':
                    print('NO DCD FILES FOUND IN CONFIG')
                coordDcd.append(line[:-1])

```

```

        line = txt.readline()
    if debug:
        print('Coordinate DCD files: {}'.format(coordDcd))

# Set coordinate max/min and binsize
def getCoordBounds(cF):
    global dims,hdims
    txt = open(cF,'r')
    line = txt.readline()
    length1 = len("MIN DISTANCE: ")
    length2 = len("MAX DISTANCE: ")
    length3 = len("BIN SIZE: ")
    length4 = len("OFFSET: ")
    length5 = len("ATOM1: ")
    length6 = len("ATOM2: ")
    length7 = len("ATOM3: ")

    global rMin, rMax, binSize, binCount, d, atom1, atom2, atom3

# scan config file for coord and bin values
while line != "END CONFIG\n":
    line = txt.readline()
    if line[:length1] == "MIN DISTANCE: ":
        rem = -1 * (len(line) - length1)
        rMin = int(line[rem:-1])
    elif line[:length2] == "MAX DISTANCE: ":
        rem = -1 * (len(line) - length2)
        rMax = int(line[rem:-1])
    elif line[:length3] == "BIN SIZE: ":
        rem = -1 * (len(line) - length3)
        binSize = float(line[rem:-1])
    elif line[:length4] == "OFFSET: ":
        rem = -1 * (len(line) - length4)
        d = float(line[rem:-1])
    elif line[:length5] == "ATOM1: ":
        rem = -1 * (len(line) - length5)
        atom1 = str(line[rem:-1])
    elif line[:length6] == "ATOM2: ":
        rem = -1 * (len(line) - length6)
        atom2 = str(line[rem:-1])
    elif line[:length7] == "ATOM3: ":
        rem = -1 * (len(line) - length7)
        atom3 = str(line[rem:-1])

# Define subset of data without solvent
def parseWater():
    # select all atoms that are not water or hydrogen
    if debug:
        print("\n--Reading DCD Data--\n\t- Parsing out WATER...\n")
    global ionsCoord,ionsForce
    global H2OCoord,H2OForce
    H2OCoord = coord.select_atoms("resname CL3")
    ionsCoord = coord.select_atoms("resname PDI")

# Initialize MD Analysis
def initMDA():
    global coord, dims, hdims, rMax, binCount, debug,force

    m = False # Switched to True if user requests an rMax value greater than the
               # system allows
    # coordinate universe
    coord = MDAnalysis.Universe(psf, coordDcd)
    dims = [coord.dimensions[0], coord.dimensions[1], coord.dimensions[2]]
    hdims = [dims[0]/2,dims[1]/2,dims[2]/2]
    rMaxLimit = np.sqrt((dims[0]**2) + (dims[1]**2) + (dims[2]**2))

```

```

if rMax > rMaxLimit:
    rMax = rMaxLimit
    m = True

binCount = int((rMax - rMin)/binSize)

if debug:
    print("--Dimensions of System--")
    print("\tTotal System Dimensions: {} A x {} A x {} A".format(dims[0], dims[1],
dims[2]))
    print("\tMin Interparticle Distance Considered: {} A".format(rMin))
    if m:
        print("\tMax Interparticle Distance Considered: {} A\tThe requested rMax
value was bigger than the simulation box size permits, and was truncated".format(
rMax))
    else:
        print("\tMax Interparticle Distance Considered: {} A".format(rMax))
        print("\tBin Size Used: {}".format(binSize))
        print("\tBin Count: {}".format(binCount))
        print("\tExcluded Volume Offset: {}".format(d))

# Truncate solvent out of the data
parseWater()

# Print log data
printLogData(debug)

# Print log data
def printLogData(d):
    if d:
        global ionsCoord
        print("--Simulation Log Info--")
        # print list of atoms being considered
        print("DCD coord universe:", len(ionsCoord), "atom(s)")
        for i in range(0, len(ionsCoord)):
            print("\t", ionsCoord[i])
        # some general log info
        print("\nNumber of time steps in coordinate trajectory:", len(coord.trajectory
))

# Iterate through all pairs of particles in all simulations,
# identifying each pair of particles, performing computations,
# and storing the results in a data set
def iterate():
    global plots,dims,hdims
    ngo=0
    nWat=len(H20Coord.atoms)
    hrMax=0.5*rMax
    nH20=np.zeros((binCount,binCount,binCount),dtype=np.int)
    pH20=np.zeros((binCount,binCount,binCount,3),dtype=np.float)
    p2H20=np.zeros((binCount,binCount,binCount,3,3),dtype=np.float)
    if debug:
        print("-- Iterating through all particle pairs in first time step to establish
pair types")
    for ts in coord.trajectory:
        # Iterate through all time steps
        dims = [coord.dimensions[0], coord.dimensions[1], coord.dimensions[2]]
        hdims = [dims[0]/2.,dims[1]/2.,dims[2]/2.]
        ngo+=1

        sys.stdout.write("Progress: {0:.2f}% Complete\r".format((float(ts.frame) /
float(len(coord.trajectory))) * 100))
        sys.stdout.flush()

# Compute radial vectors

```

```

    if ts.frame <= 0:
# Compute solute vectors
    axes=np.zeros((3,3),dtype=float) # these dimensions because each atom
    position is a 3 element sequence
    sel1 = atom1 # "resid 1 and name "+atom1
    sel2 = atom2
    if atom3 != '.': # NOTE: for LJ sphere dimer simulations use atom3 = . in
the config file
        sel3 = atom3

    sel1_univ = coord.select_atoms(sel1)
    sel2_univ = coord.select_atoms(sel2)
    if atom3 != ".":
        sel3_univ = coord.select_atoms(sel3)
# Atom positions
    if atom3 != '.':
        atom1_pos = sel1_univ.atoms[0].position
        atom2_pos = sel2_univ.atoms[0].position
        atom3_pos = sel3_univ.atoms[0].position
    else:
        atom1_pos = sel1_univ.atoms[0].position
        atom2_pos = sel1_univ.atoms[1].position
        atom3_pos = np.zeros(3, dtype=float)
# Find 3 axes of solute
    r1 = atom2_pos-atom1_pos
    r1 /= np.sqrt(np.dot(r1,r1))
    t1 = atom3_pos-atom1_pos
    r3 = np.cross(r1,t1)
    r3 /= np.sqrt(np.dot(r3,r3))
    r2 = np.cross(r3,r1)
    r2 /= np.sqrt(np.dot(r2,r2))
# Define 3 axes of solute
    axes[0] = r1
    axes[1] = r2
    axes[2] = r3

    rCen=np.zeros(3)
    mtot=0
    for a in ionsCoord:
        rCen+=a.position*a.mass
        mtot+=a.mass
    rCen=rCen/mtot
    if debug:
        print("-- Printing the .crd file")
    outCrd = open(outname+".p2.crd", 'w')
    ntyp=0
    i=-1
    typlist=[]
    atyp=np.zeros(len(ionsCoord),dtype=np.int)
    for a in ionsCoord:
        i+=1
        inew=1
        for jtyp in typlist:
            if a.type_index == jtyp[1]:
                atyp[i]=jtyp[0]
                inew=0
        if inew == 1:
            ntyp+=1
            typlist.append([ntyp,a.type_index])
            atyp[i]=ntyp
    i=0
    outCrd.write("{:4d} {:4d}\n".format(len(ionsCoord),ntyp))
    for a in ionsCoord:
        rNew = np.dot(axes,a.position-rCen)

```

```

        outCrD.write("{:3d} {:12.6f} {:12.6f} {:12.6f}\n".format(atyp[i],rNew
[0],rNew[1],rNew[2])) # write rotated solute coordinates
        #outCrD.write("{:3d} {:12.6f} {:12.6f} {:12.6f}\n".format(atyp[i],a.
position[0]-rCen[0],a.position[1]-rCen[1],a.position[2]-rCen[2]))
        i+=1
    outCrD.close()

    # Calculate x,y,z bin
    #rWat=H2OCoord.atoms[np.arange(1,nWat,5)].positions-rCen # original
    rWat = H2OCoord.atoms[np.arange(1,nWat,5)].positions - rCen # C center
    pWat = H2OCoord.atoms[np.arange(1,nWat,5)].positions - H2OCoord.atoms[np.
arange(0,nWat,5)].positions # points from H --> C
    np.divide(pWat, np.sqrt(np.einsum("ij,ij->i",pWat,pWat))[:,None], out=pWat) #
normalize by magnitude, make them all unit vectors
    rWat += d*pWat # center of excluded volume. Add because p points from H-->C
    rWat = np.einsum('ij,kj->ki',axes,rWat) # XXX rotate into solute axes
    pWat = np.einsum('ij,kj->ki',axes,pWat) # XXX rotate into solute axes
    ir = np.where(np.any(abs(rWat)>hrMax,axis=1))
    rWat = np.delete(rWat,ir,axis=0)
    pWat = np.delete(pWat,ir,axis=0)
    pWat2=np.einsum('ij,ik->ijk',pWat,pWat)
    ix=np.ndarray.astype((rWat[:]+hrMax)/binSize,np.int)
    np.add.at(nH20,tuple(ix.T),1)
    np.add.at(pH20,tuple(ix.T),pWat)
    np.add.at(p2H20,tuple(ix.T),pWat2)

#p0=np.linalg.norm(pWat[0])
#print(p0)
#p02=p0**2
dxH20=1/(binSize**3*0.00750924*ngo)
#np.divide(pH20,p0*nH20[:, :, :, None],out=pH20,where=nH20[:, :, :, None]!=0)
np.divide(pH20,nH20[:, :, :, None],out=pH20,where=nH20[:, :, :, None]!=0)
#np.divide(p2H20,p02*nH20[:, :, :, None, None],out=p2H20,where=nH20[:, :, :, None, None
]!=0)
print("Writing output file ==> {}.p2.gr3".format(outname))
outFile = open(outname+".p2.gr3", 'w')
outFile.write("# 1: X Axis # 8: Counts\n")
outFile.write("# 2: Y Axis\n")
outFile.write("# 3: Z Axis\n")
outFile.write("# 4: g(r)\n")
outFile.write("# 5: px(r)\n")
outFile.write("# 6: py(r)\n")
outFile.write("# 7: pz(r)\n")
#outFile.write("{:12d}\n".format(binCount*binCount*binCount))
for i in range(binCount):
    for j in range(binCount):
        for k in range(binCount):
            outFile.write("{:7.3f} {:7.3f} {:7.3f} {:18.12f} {:18.12f} {:18.12f}
{:18.12f} {:8d}\n".format((i+0.5)*binSize-hrMax,(j+0.5)*binSize-hrMax,(k+0.5)*
binSize-hrMax,nH20[i][j][k]*dxH20,pH20[i][j][k][0],pH20[i][j][k][1],pH20[i][j][k
][2],nH20[i][j][k]))
#outFile.write("# 1: X Axis # 8: px(r) Off-Diagonal Variance\n")
#outFile.write("# 2: Y Axis # 9: py(r) Off-Diagonal Variance\n")
#outFile.write("# 3: Z Axis # 10: pz(r) Off-Diagonal Variance\n")
#outFile.write("# 4: g(r) # 11: On-Diagonal Variance\n")
#outFile.write("# 5: px(r) # 12: On-Diagonal Variance\n")
#outFile.write("# 6: py(r) # 13: Counts\n")
#outFile.write("# 7: pz(r)\n")
#outFile.write("{:7.3f} {:7.3f} {:7.3f} {:18.12f} {:18.12f} {:18.12f}
{:18.12f} {:18.12f} {:18.12f} {:18.12f} {:18.12f} {:18.12f} {:8d}\n".format((i+0.5)
*binSize-hrMax,(j+0.5)*binSize-hrMax,(k+0.5)*binSize-hrMax,nH20[i][j][k]*dxH20,pH20
[i][j][k][0],pH20[i][j][k][1],pH20[i][j][k][2],p2H20[i][j][k][0][0],p2H20[i][j][k
][0][1],p2H20[i][j][k][0][2],p2H20[i][j][k][1][1],p2H20[i][j][k][1][2],nH20[i][j][k
]))
outFile.close()

```



```

#####

# main program
def main():

    # access global var for config file
    global configFile, pdb
    start = time.time()

    # Read config setting for debug mode
    setDebug(configFile)

    # Get name of PSF file from config file
    getPsf(configFile)

    # Get name of OUTPUT file from config file
    getOut(configFile)

    # Get names of Coord DCD files from config file
    getCoordDCDs(configFile)

    # Define coordinate min/max and bin size
    getCoordBounds(configFile)

    # Initialize MD Analysis
    initMDA()

    # Iterate over time steps, and perform MD calculations
    iterate()

    end = time.time()
    t = end - start
    print("\nTotal running time: {:.2f} sec".format(t))

# Main program code
main()

```

## 2.3 Fitting IS-SPA 3D Histogram $\chi^2$ : SPA.poisson.omp.rc16.sym.f

This code was written by Dr. Peter "Rex" Lake with minor alterations by myself.

```

CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
PROGRAM fit_gr
    include 'omp_lib.h'
    integer nmax,mpts,hmax
    parameter(nmax=96,mpts=500000000,hmax=160)
    double precision y(mpts),x(3,mpts),wy(mpts)
    double precision rmin(nmax)
    double precision xpos(3,nmax)
    double precision g(nmax*hmax)
    double precision g2(nmax*hmax),g3(nmax*hmax)
    integer ni(nmax*hmax)
    integer imap(nmax*hmax)
    double precision nc(nmax*hmax,nmax*hmax)
    double precision nc2(nmax*hmax,nmax*hmax)
    integer ityp(nmax),inow(nmax)
    double precision xnow,xmax,y0,e0,s0,p1,p2,p3, binSize
    integer i,j,jmin,npts,ntyp,npar,k,jp,kp,nmat,igo,ncpu,jt
    integer ncmts
    character*128 crdF, gr3F, outF

C
    call omp_set_dynamic(.false.)
C !xxx

```

```

open(20,file='poisson1.inp',status='old')
read(20,*)
read(20,'(a)') crdF
read(20,*)
read(20,'(a)') gr3F
read(20,*)
read(20,'(a)') outF
read(20,*)
read(20,*) y0
read(20,*)
read(20,*) ncpu
read(20,*)
read(20,*) ncmts
read(20,*)
read(20,*) npts
read(20,*)
read(20,*) binSize
close(20)
write(6,*) 'crd File: ', crdF
write(6,*) 'gr3 File: ', gr3F
write(6,*) 'Output File: ', outF
write(6,*) 'One Count Density: ', y0
write(6,*) 'CPUs: ', ncpu
write(6,*) 'Comment Lines: ', ncmts
write(6,*) 'Data Lines (gr3): ', npts
write(6,*) 'Bin Size: ', binSize
C !xxx
C read in the molecule coordinates and types
open(20,FILE=crdF,STATUS='old')
read(20,*) npar,ntyp
print*, 'Number of Atoms and Types ==> ', npar, ntyp
do i=1,npar
  read(20,*) ityp(i),xpos(1,i),xpos(2,i),xpos(3,i)
enddo
close(20)
C y0 = is the density value of a single count
e0=dlog(y0)-0.577215665d0
s0=3.1415926535898d0**2/6.d0
C
do i=1,nmax*hmax
  do j=1,nmax*hmax
    nc(j,i)=0.d0
  enddo
  g(i)=0.d0
  ni(i)=0
enddo
C read in the distribution function
open(20,FILE=gr3F,STATUS='old')
C read(20,*) npts
do i = 1, ncmts !xxx
  read(20,*)
end do
do i=1,npts
  read(20,*) x(1,i),x(2,i),x(3,i),y(i) !,p1,p2,p3
  jp=int(y(i)/y0+0.1d0)
  y(i)=e0
  wy(i)=s0
  do j=1,jp
    y(i)=y(i)+1.d0/dble(j)
    wy(i)=wy(i)-1.d0/dble(j*j)
  enddo
  wy(i)=1.d0/wy(i)
  do j=1,npar
    jt=ityp(j)
    xnow=(x(1,i)-xpos(1,j))**2+

```

```

&          (x(2,i)-xpos(2,j))**2+
&          (x(3,i)-xpos(3,j))**2
      if(xnow.lt.256.d0) then
        k=int(dsqrt(xnow)/0.1d0)+1
        k=hmax*(jt-1)+k
        ni(k)=ni(k)+int(wy(i)+0.1d0)
      endif
    enddo
  enddo
close(20)

C
write(6,*) 'finding rmin'
C
do i=1,ntyp
C
  rmin(i)=1000.d0
C
  enddo
C
do i=1,npts
C
  if(wy(i).gt.1.d0/s0+0.001d0) then
C
    do j=1,npar
C
      jt=ityp(j)
C
      xnow=(x(1,i)-xpos(1,j))**2
C
      &          +(x(2,i)-xpos(2,j))**2
C
      &          +(x(3,i)-xpos(3,j))**2
C
      if(xnow.lt.rmin(jt)) rmin(jt)=xnow
C
    enddo
C
  endif
C
  enddo
C
do i=1,ntyp
C
  write(6,*) rmin(i)
C
  enddo
C
nmat=0 !debug
do i=1,ntyp
  jp=(i-1)*hmax+1
  j=1
  do while(ni(jp).le.1)
    jp=jp+1
    if (ni(jp).eq.1) print*, 'shit',i
    j=j+1
  enddo
  nmat=nmat+hmax-j+1
  rmin(i)=dble(j-1)*0.1d0
  rmin(i)=rmin(i)*rmin(i)
enddo
print*, 'nmat:', nmat

C
write(0,*) 'looping over grid'
do i=1,npts
  if(wy(i).gt.0.1d0) then
    igo=1
    j=1
    jt=ityp(j)
    do while (igo.gt.0)
      xnow=(x(1,i)-xpos(1,j))**2
&          +(x(2,i)-xpos(2,j))**2
&          +(x(3,i)-xpos(3,j))**2
      if(xnow.gt.rmin(jt)) then
        xnow=dsqrt(xnow)
        inow(j)=int(xnow/0.1d0)+1
        if(inow(j).le.hmax) then
          inow(j)=hmax*(jt-1)+inow(j)
          igo=igo+1
        else
          inow(j)=0
        endif
      else
        igo=0
      endif
    enddo
  else
    igo=0
  endif
enddo

```

```

endif
if(j.eq.npar) then
  if (igo.gt.1) then
    igo=-1
  else
    igo=0
  endif
else
  j=j+1
  jt=ityp(j)
endif
enddo

C

if(igo.eq.-1) then
do j=1,npar
  jp=inow(j)
  if(jp.gt.0) then
    g(jp)=g(jp)+wy(i)*y(i)
    do k=1,npar
      kp=inow(k)
      if(kp.gt.0) then
        nc(kp,jp)=nc(kp,jp)+wy(i)
      endif
    enddo
  endif
enddo
endif
endif
enddo

C

write(0,*) 'reordering matrix'
nmat=0
do i=1,hmax*nmax
  if(nc(i,i).gt.0.5d0) then
C
    if(mod(i,hmax).ne.120.or.i.eq.120) then
      nmat=nmat+1
      if(nmat.ne.i) then
        do j=1,hmax*nmax
          nc(j,nmat)=nc(j,i)
        enddo
        do j=1,hmax*nmax
          nc(nmat,j)=nc(i,j)
        enddo
        g(nmat)=g(i)
      endif
      imap(i)=nmat
C
    else
C
      nmat=nmat+1
C
      do j=1,hmax*nmax
C
        nc(nmat,j)=0.d0
C
      enddo
C
      nc(nmat,nmat)=1.d0
C
      jp=imap(i-hmax)
C
      nc(nmat,jp)=-1.d0
C
      g(nmat)=0.d0
C
      imap(i)=nmat
C
      imap(i)=0
C
    endif
  else
    imap(i)=0
  endif
enddo

C

write(0,*) 'total:',nmat

```



```

                if(abs(mat(j,k)).ge.big2(tid)) then
                    big2(tid)=abs(mat(j,k))
                    irow2(tid)=j
                    icol2(tid)=k
                endif
            endif
        enddo
    endif
enddo
C$OMP enddo
C$OMP end parallel
big=big2(1)
irow=irow2(1)
icol=icol2(1)
do j=2,ncpu
    if(big2(j).gt.big) then
        big=big2(j)
        irow=irow2(j)
        icol=icol2(j)
    endif
enddo
ipiv(icol)=ipiv(icol)+1
C
    if(irow.ne.icol) then
C$OMP parallel do schedule(static) num_threads(ncpu) default(none)
C$OMP& private(j,dum)
C$OMP& shared(nmat,mat,irow,icol)
        do j=1,nmat
            dum=mat(irow,j)
            mat(irow,j)=mat(icol,j)
            mat(icol,j)=dum
        enddo
C$OMP end parallel do
        dum=val(irow)
        val(irow)=val(icol)
        val(icol)=dum
    endif
    indxr(i)=irow
    indxc(i)=icol
    pivinv=1.d0/mat(icol,icol)
    mat(icol,icol)=1.d0
C$OMP parallel do schedule(static) num_threads(ncpu) default(none)
C$OMP& private(j,dum)
C$OMP& shared(nmat,mat,icol,pivinv)
        do j=1,nmat
            mat(icol,j)=mat(icol,j)*pivinv
        enddo
C$OMP end parallel do
        val(icol)=val(icol)*pivinv
C
C$OMP parallel do schedule(guided) num_threads(ncpu) default(none)
C$OMP& private(j,dum,k)
C$OMP& shared(nmat,mat,icol,val)
        do j=1,nmat
            if(j.ne.icol) then
                dum=mat(j,icol)
                mat(j,icol)=0.d0
                do k=1,nmat
                    mat(j,k)=mat(j,k)-mat(icol,k)*dum
                enddo
                val(j)=val(j)-val(icol)*dum
            endif
        enddo
C$OMP end parallel do
enddo

```

```

C
C      do i=nmat,1,-1
C          if(indxr(i).ne.indxc(i)) then
C              do j=1,nmat
C                  dum=mat(j,indxr(i))
C                  mat(j,indxr(i))=mat(j,indxc(i))
C                  mat(j,indxc(i))=dum
C              enddo
C          endif
C      enddo
C
C      return
C      end
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

```

## 2.4 Poisson Regression Fitting Code: SPA.poisson2.omp.rc16.sym.f

This code was written by Dr. Peter "Rex" Lake with minor alterations by myself.

```

CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
PROGRAM fit_gr
    include 'omp_lib.h'
    integer nmax,mpts,hmax
    double precision boxSize
    parameter(nmax=96,mpts=500000000,hmax=160,boxSize=25.d0)
    double precision y,hy0 ,y0
    double precision rmin(nmax)
    double precision x(3,nmax)
    double precision g(nmax*hmax)
    double precision a(nmax*hmax,nmax*hmax)
    double precision b(nmax*hmax),b0(nmax*hmax)
    integer ityp(nmax),inow(nmax),imin(2,nmax)
    double precision xnow,xmax,p1,p2,p3,x1,x2,x3
    double precision dnow,dmax,gnow,r0,r2
    double precision lnp, binSize
    integer ix,iy,iz,ip,ir,istop,ncell
    integer i,j,jmin,npts,ntyp,npar,k,jp,kp,nmat,igo,ncpu,ngo,jt
    character*128 crdF, gr3F, outF, pos1F
    integer ncmts
    integer ib(nmax*hmax)
    integer bins
C
C      call omp_set_dynamic(.false.)
C !xxx
    open(20,file='poisson2.inp',status='old')
    read(20,*)
    read(20,'(a)') crdF
    read(20,*)
    read(20,'(a)') gr3F
    read(20,*)
    read(20,'(a)') pos1F
    read(20,*)
    read(20,'(a)') outF
    read(20,*)
    read(20,*) y0
    read(20,*)
    read(20,*) ncpu
    read(20,*)
    read(20,*) ncmts
    read(20,*)
    read(20,*) npts
    read(20,*)
    read(20,*) binSize

```

```

close(20)
write(6,*) 'crd File: ', crdF
write(6,*) 'gr3 File: ', gr3F
write(6,*) 'Poisson1 File: ', pos1F
write(6,*) 'Output File: ', outF
write(6,*) 'One Count Density: ', y0
write(6,*) 'CPUs: ', ncpu
write(6,*) 'Comment Lines: ', ncmts
write(6,*) 'Data Lines (gr3): ', npts
write(6,*) 'Bin Size: ', binSize
C !xxx

C Half the density reported in a cell with one count.
hy0=y0*0.5d0
C

C read in the molecule coordinates and types
open(20,FILE=crdF,STATUS='old')
read(20,*) npar,ntyp
print*, 'Number of Atoms and Types ==> ', npar, ntyp
do i=1,npar
  read(20,*) ityp(i),x(1,i),x(2,i),x(3,i)
enddo
close(20)
C read in previous fit - determine imin
open(20,FILE=pos1F,STATUS='unknown')
nmat=0
do i=1,ntyp
  imin(1,i)=nmat
  do j=1,hmax
    read(20,*) xnow,gnow
    if(dabs(gnow).lt.1.d-6) then
      imin(2,i)=j
      rmin(i)=xnow
    else
      nmat=nmat+1
      g(nmat)=gnow
    endif
  enddo
  rmin(i)=rmin(i)+0.05d0
  rmin(i)=rmin(i)*rmin(i)
enddo
close(20)
C
do i=1,nmat
  b0(i)=0.d0
enddo
C read in the distribution function
write(0,*) 'reading g(r)'
open(20,FILE=gr3F,STATUS='old')
C read(20,*) npts
do i = 1, ncmts !xxx
  read(20,*)
end do
do i=1,npts
  read(20,*) x1,x2,x3,y !,p1,p2,p3
  if(y.gt.hy0) then
    igo=1
    j=1
    do while (igo.gt.0)
      jt=ityp(j)
      r2=(x1-x(1,j))**2+(x2-x(2,j))**2+(x3-x(3,j))**2
      if(r2.lt.256.d0) igo=igo+1
      if(r2.lt.rmin(jt)) igo=0
      j=j+1
    enddo
  enddo
enddo

```



```

        if((igo.ne.0).and.(j.gt.npar)) igo=-igo
    end do
    if(igo.lt.-1) then
        do j=1,npar
            jt=ityp(j)
            r2=(x1-x(1,j))**2+(x2-x(2,j))**2+(x3-x(3,j))**2
            if (r2.lt.256.d0) then
                r0=dsqrt(r2)
                ir=int(r0/0.1d0)+1
                k=imin(1,jt)+ir-imin(2,jt)
                if (k==0) then !debug
                    print*, 'k is 0!', jt,imin(1,jt),ir,imin(2,jt),r0
                    flush(6)
                end if
                b0(k)=b0(k)-y
            endif
        enddo
    endif
enddo
close(20)
do i=1,nmat
    if (b0(i).eq.0) print*, i, nmat, npts
end do
print*, 'Donee'

```

C

```

istop=0
do while (istop.eq.0)
    do i=1,nmat
        do j=1,nmat
            a(j,i)=0.d0
        enddo
        b(i)=b0(i)
        ib(i)=0
    enddo

```

C

```

write(0,*) 'looping over grid'
bins=int(2*boxSize/binSize+0.1d0)
print*, 'bins of box:', bins
do ix=1,bins
    x1=(dble(ix)-0.5d0)*binSize-boxSize
    do iy=1,bins
        x2=(dble(iy)-0.5d0)*binSize-boxSize
        do iz=1,bins
            x3=(dble(iz)-0.5d0)*binSize-boxSize

```

C

```

        igo=1
        gnow=0.d0
        j=1
        jt=ityp(j)
        do while (igo.gt.0)
            xnow=(x(1,j)-x1)**2+(x(2,j)-x2)**2+(x(3,j)-x3)**2
            if(xnow.gt.rmin(jt)) then
                xnow=dsqrt(xnow)
                ip=int(xnow/0.1d0)+1
                if(ip.le.hmax) then
                    ip=imin(1,jt)+ip-imin(2,jt)
                    gnow=gnow+g(ip)
                    inow(j)=ip
                    igo=igo+1
                else
                    inow(j)=0
                endif
            else
                inow(j)=0
            endif
        enddo
    else
        inow(j)=0
    endif
enddo

```

```

        igo=0
    endif
    if(j.eq.npar) then
        if (igo.gt.1) then
            igo=-1
        else
            igo=0
        endif
    else
        j=j+1
        jt=ityp(j)
    endif
enddo

C
    if(igo.eq.-1) then
        gnow=dexp(gnow)
        do j=1,npar
            jp=inow(j)
            if(jp.gt.0) then
                b(jp)=b(jp)+gnow
                ib(jp)=ib(jp)+1
                do k=1,npar
                    kp=inow(k)
                    if(kp.gt.0) then
                        a(kp,jp)=a(kp,jp)+gnow
                    endif
                enddo
            endif
        enddo
    endif
enddo
enddo
enddo
enddo

C
write(0,*) 'total:',nmat
call inv(a,b,nmat,ncpu)
dmax=0.d0
do i=1,nmat
    dnow=dabs(b(i))
    if(dnow.gt.dmax) dmax=dnow
    g(i)=g(i)-b(i)
enddo

C
write(0,*) dmax
if (dmax.lt.0.01d0) istop=1

C
open(20,FILE=outF,STATUS='unknown')
jp=0
do i=1,ntyp
    do j=1,imin(2,i)
        xnow=(j-0.5d0)*0.1d0
        write(20,*) xnow,0.d0,i,-1
    enddo
    do j=imin(2,i)+1,hmax
        jp=jp+1
        xnow=(j-0.5d0)*0.1d0
        write(20,*) xnow,g(jp),i,ib(jp)
    enddo
enddo
close(20)
enddo

C
lnp=0.d0
C
write(0,*) 'reading g(r)'

```

```

C      write(fname,887) lett(ngo)
C      open(20,FILE=fname,STATUS='old')
C      read(20,*) npts
C      do i=1,npts
C          read(20,*) x1,x2,x3,y,p1,p2,p3,wy
C
C          igo=1
C          gnow=g0
C          j=1
C          jt=ityp(j)
C          do while (igo.gt.0)
C              xnow=(x(1,j)-x1)**2+(x(2,j)-x2)**2+(x(3,j)-x3)**2
C              if(xnow.gt.rmin(jt)) then
C                  xnow=dsqrt(xnow)
C                  ip=int(xnow/0.1d0)+1
C                  if(ip.le.hmax) then
C                      ip=imin(1,jt)+ip-imin(2,jt)
C                      gnow=gnow+g(ip)
C                      inow(j)=ip
C                      igo=igo+1
C                  else
C                      inow(j)=0
C                  endif
C              else
C                  igo=0
C              endif
C              if(j.eq.npar) then
C                  if (igo.gt.1) then
C                      igo=-1
C                  else
C                      igo=0
C                  endif
C              else
C                  j=j+1
C                  jt=ityp(j)
C              endif
C          enddo
C
C      if(igo.eq.-1) then
C          ncell=ncell+1
C          lnp=lnp+lgamma(wy+1.d0)+dexp(gnow)-wy*gnow
C
C      endif
C
C      enddo
C      close(20)
C      write(0,*) '-ln(p)=' ,lnp
C      write(0,*) 'ncell=' ,ncell
C
C
C      return
C      end
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

```

```

subroutine inv(mat, val, nmat, ncpu)
include 'omp_lib.h'
integer nmax, hmax
parameter(nmax=96, hmax=160)
double precision mat(nmax*hmax, nmax*hmax)
double precision val(nmax*hmax)
integer irow2(20), icol2(20)
double precision big2(20)
integer ipiv(nmax*hmax)
integer i, j, k, irow, icol, nmat, imat, ncpu, tid
integer indx(nmax*hmax), indxr(nmax*hmax)

```

```

        double precision big,pivinv,dum
C
    do i=1,nmat
        ipiv(i)=0
    enddo
C
    do i=1,nmat
C
        write(0,*) i
        do j=1,ncpu
            big2(j)=0.d0
        enddo
C$OMP parallel num_threads(ncpu) default(none)
C$OMP& private(j,k,tid)
C$OMP& shared(i,nmat,ipiv,mat,big2,irow2,icol2)
        tid=omp_get_thread_num()+1
C$OMP do schedule(guided)
        do j=1,nmat
            if(ipiv(j).eq.0) then
                do k=1,nmat
                    if(ipiv(k).eq.0) then
                        if(abs(mat(j,k)).ge.big2(tid)) then
                            big2(tid)=abs(mat(j,k))
                            irow2(tid)=j
                            icol2(tid)=k
                        endif
                    endif
                enddo
            endif
        enddo
C$OMP enddo
C$OMP end parallel
        big=big2(1)
        irow=irow2(1)
        icol=icol2(1)
        do j=2,ncpu
            if(big2(j).gt.big) then
                big=big2(j)
                irow=irow2(j)
                icol=icol2(j)
            endif
        enddo
        ipiv(icol)=ipiv(icol)+1
C
        if(irow.ne.icol) then
C$OMP parallel do schedule(static) num_threads(ncpu) default(none)
C$OMP& private(j,dum)
C$OMP& shared(nmat,mat,irow,icol)
            do j=1,nmat
                dum=mat(irow,j)
                mat(irow,j)=mat(icol,j)
                mat(icol,j)=dum
            enddo
C$OMP end parallel do
            dum=val(irow)
            val(irow)=val(icol)
            val(icol)=dum
        endif
        indxr(i)=irow
        indxk(i)=icol
        pivinv=1.d0/mat(icol,icol)
        mat(icol,icol)=1.d0
C$OMP parallel do schedule(static) num_threads(ncpu) default(none)
C$OMP& private(j,dum)
C$OMP& shared(nmat,mat,icol,pivinv)
        do j=1,nmat

```



```

real(kind=dp),allocatable :: g1D(:,,:), fLJr1D(:,,:), fCr1D(:,,:), g1D2(:,,:), fLJr1D2
(:,,:), fCr1D2(:,,:)
real(kind=dp) :: histDistStepSize, histCosThStepSize, histPhiStepSize
integer :: histDistBins, histCosThBins, histPhiBins

end module histData

! data from the config file.
module cfgData
  use prec
  use constants
  real(kind=dp),allocatable :: x_axis(:), z_axis(:), R_axis(:), fAvg(:,,:), u_dir(:,,:)
  real(kind=dp) :: RStepSize, xzStepSize, R_min, R_max, xz_range, cfgCosThStepSize,
  cfgPsiStepSize, T, cut, offset, soluteChg(2), radius
  character(len=8) :: c_explicit_R
  integer :: cfgRBins, cfgCosThBins, cfgPhiBins, cfgPsiBins, nThreads
!
  integer :: xBins, zBins
  real(kind=dp) :: density = 0.00750924_dp ! numerical density of chloroforms per
  Angstrom**3
  real(kind=dp) :: cosTh_max = 1_dp
  real(kind=dp) :: cosTh_min = -1_dp
  real(kind=dp) :: phi_max = pi/3_dp
  real(kind=dp) :: phi_min = 0_dp
  real(kind=dp) :: psi_max = 2_dp*pi
  real(kind=dp) :: psi_min = 0_dp
end module cfgData

! data for calculating cosTh value.
module angleData
  use prec
  real(kind=dp),allocatable :: sinThetaLF(:), cosThetaLF(:), sinPhiLF(:), cosPhiLF(:)
  , sinPsiLF(:), cosPsiLF(:)
  real(kind=dp) :: rSolv1(3), rSolv2(3), rSolvn(2), cosTh(2), phi(2), sSolv1(3),
  tSolv1(3), sSolvln, tSolvln
!$omp THREADPRIVATE( rSolv1, rSolv2, rSolvn, sSolv1, sSolvln, tSolv1, tSolvln,
  cosTh, phi )
end module angleData

! testing arrays for force and g(r)
module ctrlData
  use prec
  real(kind=dp),allocatable :: frcSPA(:, :, :, :), grSPA(:, :), explicitDist(:)
  integer :: crdLines
  logical :: explicit_R
end module ctrlData

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!! Main Program !!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

program compute_avgForce
  use prec
  implicit none
  character(len=128) :: histFile, cfgFile, outFile
  real(kind=dp) :: omp_get_wtime, ti, tf, seconds
  integer :: hours, minutes

  ti = omp_get_wtime()

  ! make list of average direct force from 'collapsed' file.
  call parse_command_line(cfgFile)

```

```

! read config file
call read_cfg(cfgFile, histFile, outFile)

! make list of average direct force from 'collapsed' file.
call make_hist_table(histFile)

! Now that we have the relevant information spline the g and f arrays along r.
call spline_hist_array

! read in LJ--LJ dist array from file
call R_list

! setup for computing the average force integral.
call setup_compute_avg_force

! compute average force integral.
call compute_avg_force

! integrate average force to get PMF.
call integrate_force

! write PMF output file
call write_output(outFile)

! Write time taken to finish calculation.
tf = omp_get_wtime()

hours = (tf-ti)/3600
minutes = mod((tf-ti),3600d0)/60
seconds = mod(mod((tf-ti),3600d0),60d0)

write(*,*) "~~~~~"
write(*,'(a,i4,a,i2,a,f6.3,a)') "Total time elapsed:  ", hours, "h ", minutes, "m
", seconds, "s"

flush(6)

end program compute_avgForce

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!! Subroutines !!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

! parse commandline for relevant files.
subroutine parse_command_line(cfgFile)
  implicit none
  character(len=128) :: cfgFile
  character(len=16) :: arg
  integer :: i
  logical :: cfgFileFlag, cfgExist

  cfgFileFlag = .false.
  cfgExist = .false.
  i=1
  do
    call get_command_argument(i,arg)
    select case (arg)

    case ('-cfg')
      i = i+1
      call get_command_argument(i,cfgFile)
      cfgFileFlag=.true.
      INQUIRE(FILE=cfgFile, EXIST=cfgExist)
      write(*,*) 'Config File:          ', cfgFile

```

```

        write(*,*) 'Config File Exists:          ', cfgExist
    case default
        write(*,*) 'Unrecognized command-line option: ', arg
        write(*,*) 'Usage: compute_avgForce.x -cfg [cfg file]'
        stop

    end select
    i = i+1
    if (i.ge.command_argument_count()) exit
end do

if (cfgFileFlag.eqv..false.) then
    write(*,*) "Must provide a cfg file using command line argument -cfg [cfg file
    name]"
    stop
end if

! 'ERROR STOP' if either file doesn't exist
if (cfgExist.eqv..false.) then
    write(*,*) 'cfg file does not exist'
    error stop
end if

flush(6)

end subroutine parse_command_line

! read python cfg file for g(r) parameters
subroutine read_cfg(cfgFile, histFile, outFile)
    use cfgData
    implicit none
    character(len=128) :: cfgFile, histFile, outFile
    character(len=256) :: line
    character(len=32) :: firstWord, sep
    integer :: ios
    logical :: outFileFlag, histFileFlag, histExist, RstepSizeFlag, xzStepSizeFlag,
    RmaxFlag, RminFlag, xzRangeFlag, &
    & thetaBinsFlag, phiBinsFlag, psiBinsFlag, c_explicit_RFlag, TFlag, cutFlag,
    radiusFlag, offsetFlag, nThreadsFlag, &
    & soluteChgFlag

    histFileFlag = .false.; histExist = .false.
    outFileFlag = .false.
    RstepSizeFlag = .false.
    xzStepSizeFlag = .false.
    c_explicit_RFlag = .false.
    RmaxFlag = .false.
    RminFlag = .false.
    xzRangeFlag = .false.
    thetaBinsFlag = .false.
    phiBinsFlag = .false.
    psiBinsFlag = .false.
    TFlag = .false.
    cutFlag = .false.
    radiusFlag = .false.
    offsetFlag = .false.
    nThreadsFlag = .false.
    soluteChgFlag = .false.

    ios = 0

    open(20, file=cfgFile)
    do while(ios>=0)
        read(20, '(a)', IOSTAT=ios) line

```



```

call split(line, '=', firstWord, sep)
if (line .ne. "") then
  if (firstWord .eq. "hist_file") then
    read(line, '(a)') histFile
    write(*,*) "Histogram File:           ", histFile
    histFileFlag = .true.
    INQUIRE(FILE=histFile, EXIST=histExist) ! check if it exists
  else if (firstWord .eq. "out_file") then
    read(line,*) outFile
    write(*,*) "Output File:           ", outFile
    outFileFlag = .true.
  else if (firstWord .eq. "RStepSize") then
    read(line,*) RStepSize
    write(*,*) "PMF Step Size:           ", RStepSize
    RstepSizeFlag = .true.
  else if (firstWord .eq. "xzStepSize") then
    read(line,*) xzStepSize
    write(*,*) "Solvent Grid Step Size:       ", xzStepSize
    xzStepSizeFlag = .true.
  else if (firstWord .eq. "explicit_R") then
    read(line,*) c_explicit_R
    write(*,*) "Use Explicit R Values:       ", c_explicit_R
    c_explicit_RFlag = .true.
  else if (firstWord .eq. "R_max") then
    read(line,*) R_max
    write(*,*) "R Maximum Value:           ", R_max
    RmaxFlag = .true.
  else if (firstWord .eq. "R_min") then
    read(line,*) R_min
    write(*,*) "R Minimum Value:           ", R_min
    RminFlag = .true.
  else if (firstWord .eq. "xz_range") then
    read(line,*) xz_range
    write(*,*) "XZ - Range:               ", xz_range
    xzRangeFlag = .true.
  else if (firstWord .eq. "theta_bins") then
    read(line,*) cfgCosThBins
    write(*,*) "Theta Bins:               ", cfgCosThBins
    thetaBinsFlag= .true.
  else if (firstWord .eq. "phi_bins") then
    read(line,*) cfgPhiBins
    write(*,*) "Phi Bins:                 ", cfgPhiBins
    phiBinsFlag= .true.
  else if (firstWord .eq. "psi_bins") then
    read(line,*) cfgPsiBins
    write(*,*) "Psi Bins:                 ", cfgPsiBins
    psiBinsFlag= .true.
  else if (firstWord .eq. "temperature") then
    read(line,*) T
    write(*,*) "Temperature (K):           ", T
    TFlag= .true.
  else if (firstWord .eq. "bicubic_cutoff") then
    read(line,*) cut
    write(*,*) "Bicubic/Bilinear Cutoff:     ", cut
    cutFlag= .true.
  else if (firstWord .eq. "solute_radius") then
    read(line,*) radius
    write(*,*) "Solute radius:             ", radius
    radiusFlag= .true.
  else if (firstWord .eq. "offset") then
    read(line,*) offset
    write(*,*) "Solvent offset distance:     ", offset
    offsetFlag= .true.
  else if (firstWord .eq. "num_threads") then
    read(line,*) nThreads

```

```

        write(*,*) "Number of Parallel Threads:      ", nThreads
        nThreadsFlag= .true.
    else if (firstWord .eq. "solute_charge") then
        read(line,*) soluteChg(1)
        soluteChg(2) = -soluteChg(1)
        write(*,*) "Solute Charges:              ", soluteChg
        soluteChgFlag= .true.
    end if
end if
end do
close(20)

if (histFileFlag.eqv..false.) then
    write(*,*) "Config file must have a 'hist_file' value"
    stop
end if
if (histExist.eqv..false.) then
    write(*,*) "Config file must point to a 'hist_file' that exists: ", histFile, "
    doesn't exist."
    stop
end if
if (outFileFlag.eqv..false.) then
    write(*,*) "Config file must have a 'out_file' value"
    stop
end if
if (RstepSizeFlag.eqv..false.) then
    write(*,*) "Config file must have a 'RStepSize' value"
    stop
end if
if (xzStepSizeFlag.eqv..false.) then
    write(*,*) "Config file must have a 'xzStepSize' value"
    stop
end if
if (outFileFlag.eqv..false.) then
    write(*,*) "Config file must have a 'out_file' value"
    stop
end if
if (c_explicit_RFlag.eqv..false.) then
    write(*,*) "Config file must have a 'explicit_R' value"
    stop
end if
if (RmaxFlag.eqv..false.) then
    write(*,*) "Config file must have a 'R_max' value"
    stop
end if
if (RminFlag.eqv..false.) then
    write(*,*) "Config file must have a 'R_min' value"
    stop
end if
if (xzRangeFlag.eqv..false.) then
    write(*,*) "Config file must have a 'xz_range' value"
    stop
end if
if (thetaBinsFlag.eqv..false.) then
    write(*,*) "Config file must have a 'theta_bins' value"
    stop
end if
if (phiBinsFlag.eqv..false.) then
    write(*,*) "Config file must have a 'phi_bins' value"
    stop
end if
if (psiBinsFlag.eqv..false.) then
    write(*,*) "Config file must have a 'psi_bins' value"
    stop
end if

```

```

if (TFlag.eqv..false.) then
  write(*,*) "Config file must have a 'temperature' value"
  stop
end if
if (cutFlag.eqv..false.) then
  write(*,*) "Config file must have a 'bicubic_cutoff' value"
  stop
end if
if (radiusFlag.eqv..false.) then
  write(*,*) "Config file must have a 'solute_radius' value"
  stop
end if
if (offsetFlag.eqv..false.) then
  write(*,*) "Config file must have a 'offset' value"
  stop
end if
if (nThreadsFlag.eqv..false.) then
  write(*,*) "Config file must have a 'num_threads' value"
  stop
end if
if (soluteChgFlag.eqv..false.) then
  write(*,*) "Config file must have a 'solute_charge' value"
  stop
end if

flush(6)

end subroutine read_cfg

! read force file and make a lookup table.
subroutine make_hist_table(histFile)
  use histData; use cfgData
  implicit none
  character(len=128) :: histFile
  character(len=512) :: line
  integer :: ios, ios2, i, j, k, nHistLines
  real(kind=dp),allocatable :: histTmp(:, :)

  ! read number of lines in histFile and allocate that many points in temporary
  ! histogram list, histTmp.
  ios = 0; nHistLines = -1
  open(20, file=histFile)
  do while(ios>=0)
    read(20, '(a)', IOSTAT=ios) line
    if (line(1:1) .ne. "#") then
      nHistLines = nHistLines + 1
    end if
  end do
  close(20)
  write(*,*) "nHistLines", nHistLines

  allocate( histTmp(19,nHistLines) )

  ! populate hist arrays
  ios = 0; i = 1
  open(20, file=histFile)
  ! read file ignoring comment lines at the beginning
  do while(ios>=0)
    read(20, '(a)', IOSTAT=ios) line
    if ((line(1:1) .ne. "#") .and. (ios .ge. 0)) then
      !
      !           r           cos(Th)           phi/3
      read(line,*) histTmp(1,i), histTmp(2,i), histTmp(3,i), &
      !           g(r)+           g(r)-
      & histTmp(4,i), histTmp(5,i), &

```

```

!      <fLJ.r>+      <fLJ.s>+      <fLJ.t>+
& histTmp(6,i), histTmp(7,i), histTmp(8,i), &
!      <fLJ.r>-      <fLJ.s>-      <fLJ.t>-
& histTmp(9,i), histTmp(10,i), histTmp(11,i), &
!      <fC.r>+      <fC.s>+      <fC.t>+
& histTmp(12,i), histTmp(13,i), histTmp(14,i), &
!      <fC.r>-      <fC.s>-      <fC.t>-
& histTmp(15,i), histTmp(16,i), histTmp(17,i), &
!      gc(r)+      gc(r)-
& histTmp(18,i), histTmp(19,i)

    i = i + 1
  end if
end do
close(20)

! Unique value determination
do i = 1, nHistLines
  if (i .eq. 1) then
    histDistBins = 1
    ios = 0; ios2 = 0
  else ! i = 2, nHistLines
    if (( histTmp(1,i) .lt. (histTmp(1,i-1)-1d-6) ) .or. ( histTmp(1,i) .gt. (
histTmp(1,i-1)+1d-6) )) then
      ! note: this statement will trigger when a value in the first column (dist
) is different than the value in the row
      ! before it.
      histDistBins = histDistBins + 1
    end if
    if (( histTmp(2,i) .gt. (histTmp(2,1)-1d-6) ) .and. ( histTmp(2,i) .lt. (
histTmp(2,1)+1d-6) ) .and. ( ios .eq. 0 ) .and. &
& ( ios2 .eq. 1 )) then
      ! note: this statement will trigger when i = histCosThBins+1 because it
finds the first repeated element
      histCosThBins = (i - 1)/histPhiBins
      ios = 1
    end if
    if (( histTmp(3,i) .gt. (histTmp(3,1)-1d-6) ) .and. ( histTmp(3,i) .lt. (
histTmp(3,1)+1d-6) ) .and. ( ios2 .eq. 0 )) then
      ! note: this statement will trigger when i = histPhiBins+1 because it
finds the first repeated element
      histPhiBins = i - 1
      ios2 = 1
    end if
  end if
end do
write(*,*) "Histogram Distance Bins:  ", histDistBins
write(*,*) "Histogram Cosine Theta Bins:  ", histCosThBins
write(*,*) "Histogram Phi Bins:          ", histPhiBins

allocate( histDist(histDistBins), histCosTh(histCosThBins), histPhi(histPhiBins), g
(2,histDistBins,histCosThBins,histPhiBins), &
& fLJr(2,histDistBins,histCosThBins,histPhiBins), fLJs(2,histDistBins,
histCosThBins,histPhiBins), &
& fLJt(2,histDistBins,histCosThBins,histPhiBins), fCr(2,histDistBins,
histCosThBins,histPhiBins), &
& fCs(2,histDistBins,histCosThBins,histPhiBins), fCt(2,histDistBins,
histCosThBins,histPhiBins), &
& gc(2,histDistBins,histCosThBins,histPhiBins) )

! populate arrays that will be used in the rest of the calculation from temp array
do i = 1, histDistBins      ! the values written out from python script are at half
-bin distances
  histDist(i) = histTmp(1,histCosThBins*histPhiBins*(i-1)+1)
end do
do i = 1, histCosThBins

```

```

        histCosTh(i) = histTmp(2,histPhiBins*(i-1)+1)
    end do
    do i = 1, histPhiBins
        histPhi(i) = histTmp(3,i)
    end do

    do i = 1, histDistBins
        do j = 1, histCosThBins
            do k = 1, histPhiBins
                g(1,i,j,k) = histTmp( 4, (i-1)*histCosThBins*histPhiBins + (j-1)*
histPhiBins + k) ! g(r,cos,phi)+ currently g
                g(2,i,j,k) = histTmp( 5, (i-1)*histCosThBins*histPhiBins + (j-1)*
histPhiBins + k) ! g(r,cos,phi)- currently g
                fLJr(1,i,j,k) = histTmp( 6, (i-1)*histCosThBins*histPhiBins + (j-1)*
histPhiBins + k) ! <fLJ.r>(r,cos,phi) +
                fLJs(1,i,j,k) = histTmp( 7, (i-1)*histCosThBins*histPhiBins + (j-1)*
histPhiBins + k) ! <fLJ.s>(r,cos,phi) +
                fLJt(1,i,j,k) = histTmp( 8, (i-1)*histCosThBins*histPhiBins + (j-1)*
histPhiBins + k) ! <fLJ.t>(r,cos,phi) +
                fLJr(2,i,j,k) = histTmp( 9, (i-1)*histCosThBins*histPhiBins + (j-1)*
histPhiBins + k) ! <fLJ.r>(r,cos,phi) -
                fLJs(2,i,j,k) = histTmp(10, (i-1)*histCosThBins*histPhiBins + (j-1)*
histPhiBins + k) ! <fLJ.s>(r,cos,phi) -
                fLJt(2,i,j,k) = histTmp(11, (i-1)*histCosThBins*histPhiBins + (j-1)*
histPhiBins + k) ! <fLJ.t>(r,cos,phi) -
                fCr(1,i,j,k) = histTmp(12, (i-1)*histCosThBins*histPhiBins + (j-1)*
histPhiBins + k) ! <fC.r>(r,cos,phi) +
                fCs(1,i,j,k) = histTmp(13, (i-1)*histCosThBins*histPhiBins + (j-1)*
histPhiBins + k) ! <fC.s>(r,cos,phi) +
                fCt(1,i,j,k) = histTmp(14, (i-1)*histCosThBins*histPhiBins + (j-1)*
histPhiBins + k) ! <fC.t>(r,cos,phi) +
                fCr(2,i,j,k) = histTmp(15, (i-1)*histCosThBins*histPhiBins + (j-1)*
histPhiBins + k) ! <fC.r>(r,cos,phi) -
                fCs(2,i,j,k) = histTmp(16, (i-1)*histCosThBins*histPhiBins + (j-1)*
histPhiBins + k) ! <fC.s>(r,cos,phi) -
                fCt(2,i,j,k) = histTmp(17, (i-1)*histCosThBins*histPhiBins + (j-1)*
histPhiBins + k) ! <fC.t>(r,cos,phi) -
                gc(1,i,j,k) = histTmp(18, (i-1)*histCosThBins*histPhiBins + (j-1)*
histPhiBins + k) ! gc(r,cos,phi) +
                gc(2,i,j,k) = histTmp(19, (i-1)*histCosThBins*histPhiBins + (j-1)*
histPhiBins + k) ! gc(r,cos,phi) +
            end do
        end do
    end do

    histDistStepSize = histDist(2) - histDist(1)
    write(*,*) "Histogram Distance Step Size:      ", histDistStepSize
    histCosThStepSize = histCosTh(2) - histCosTh(1)
    write(*,*) "Histogram Cosine Theta Step Size:      ", histCosThStepSize
    histPhiStepSize = histPhi(2) - histPhi(1)
    write(*,*) "Histogram Phi Step Size:          ", histPhiStepSize

    flush(6)

end subroutine make_hist_table

! spline the r dimension of each theta phi stack and then average over phi for 2D
subroutine spline_hist_array
    use constants; use functions; use histData; use cfgData; use idealSolv
    implicit none
    integer :: ia, ir, ith, iphi, imin, igo, igo1, igo2
    real(kind=dp) :: norm_factor, boltz, boltz_sum
    real(kind=dp), allocatable :: idealHist(:,:,:,,:), idealHist2D(:,:,:,,:), u02D
(:,:), idealHist1D(:,:,:), u01D(:)

```

```

integer, allocatable      :: ispline(:, :), ispline2D(:)
integer :: ispline1D
!real(kind=dp)          :: xx, yy      !debug
!integer :: rTmp !debug

write(*,*) 'Editing input histogram arrays with ideal arrays in 3D...'

! Calculate a 4D array idealHist(lj+/lj-,g/f,r,th,phi)
allocate( idealHist(2,13,histDistBins,histCosThBins,histPhiBins), ispline(
histCosThBins,histPhiBins), ispline2D(histCosThBins) )
idealHist = 0_dp; ispline = 0_dp
call ideal_CL3(histDistBins,histDistStepSize,histCosThBins,cosTh_min,cosTh_max,
histPhiBins,phi_min,phi_max,radius,offset,T, &
& soluteChg, idealHist)

! Spline the log(g) and force arrays using ideal values for the slopes at small r.
This populates the second derivative arrays.
! This requires ideal values that have been averaged over phi.
allocate( idealHist2D(2,9,histDistBins,histCosThBins) )
idealHist2D = 0_dp
call ideal_3D_to_2D(idealHist,histDistBins,histCosThBins,histPhiBins, idealHist2D)

allocate( idealHist1D(2,5,histDistBins) )
idealHist1D = 0_dp
call ideal_2D_to_1D(idealHist2D,histDistBins,histCosThBins, idealHist1D)

! Allocate explicit data arrays for 3D -> 2D transformation
allocate( g2D(2,histDistBins,histCosThBins), fLJr2D(2,histDistBins,histCosThBins),
fLJs2D(2,histDistBins,histCosThBins), &
& fCr2D(2,histDistBins,histCosThBins), fCs2D(2,histDistBins,histCosThBins), u02D
(histDistBins,histCosThBins) )

g2D = 0_dp; fLJr2D = 0_dp; fLJs2D = 0_dp; fCr2D = 0_dp; fCs2D = 0_dp; u02D = 0_dp

! Allocate explicit data arrays for 2D -> 1D transformation
allocate( g1D(2,histDistBins), fLJr1D(2,histDistBins), fCr1D(2,histDistBins), u01D(
histDistBins) )
g1D = 0_dp; fLJr1D = 0_dp; fCr1D = 0_dp; u01D = 0_dp

allocate( g1D2(2,histDistBins), fLJr1D2(2,histDistBins), fCr1D2(2,histDistBins) )
g1D2 = 0_dp; fLJr1D2 = 0_dp; fCr1D2 = 0_dp

! Edit the input hist arrays to more smoothly transition to +/- infinity with the
help of idealHist.
do ia = 1, 2 ! which solute
do ith = 1, histCosThBins
do iphi = 1, histPhiBins
imin = 0
! Normalization factor for each theta phi array.
norm_factor = gc(ia,histDistBins,ith,iphi)/(g(ia,histDistBins,ith,iphi)*4*
pi*histDist(histDistBins)**2)
! Find the first non-zero g(r) bin for each theta/phi array and set 'imin'
to that 'ir' index
find: do ir = 1, histDistBins
if (g(ia,ir,ith,iphi).gt.1d-6) then
imin = ir
exit find
end if
end do find

! Note: Add the ideal values to bins with no sampling. And half counts to
bins that probably should have had sampling.
igo = 0

```

```

do ir = histDistBins, 1, -1
  if (ir.ge.imin) then
    if (g(ia,ir,ith,iphi).gt.1d-6) then
      g(ia,ir,ith,iphi) = log(g(ia,ir,ith,iphi)) ! g is ln(g) now
    else ! note: This is a zero bin where there probably should have
      been something. So put a half count in.
      g(ia,ir,ith,iphi) = log(real(0.5,dp)/(4*pi*histDist(ir)**2)/
norm_factor)

      fLJr(ia,ir,ith,iphi) = idealHist(ia, 2,ir,ith,iphi)
      fLJs(ia,ir,ith,iphi) = idealHist(ia, 3,ir,ith,iphi)
      fLJt(ia,ir,ith,iphi) = idealHist(ia, 4,ir,ith,iphi)
      fCr(ia,ir,ith,iphi) = idealHist(ia, 8,ir,ith,iphi)
      fCs(ia,ir,ith,iphi) = idealHist(ia, 9,ir,ith,iphi)
      fCt(ia,ir,ith,iphi) = idealHist(ia,10,ir,ith,iphi)
    end if
    else if (ir.lt.imin) then ! .lt.imin ==> in the region of no sampling.
Set the FE (log(g)) to the direct energy
! shifted by a constant energy term , the difference between the
last sampled indirect and direct energies.
! ln(g(r<r0)) = -u_dir(r)/T - ( u_pmf(r0)/T - u_dir(r0)/T )
! ln(g(r<r0)) = -u_dir(r)/T + ln(g(r0)) - u_dir(r0)/T
g(ia,ir,ith,iphi) = ( -idealHist(ia,1,ir,ith,iphi) + idealHist(ia,1,
imin,ith,iphi) ) + g(ia,imin,ith,iphi) ! ln(g)
if ((g(ia,ir,ith,iphi).lt.cut).and.(igo.eq.0)) then ! the largest r
to go past the cutoff
  ispline(ith,iphi) = ir
  igo = 1
end if
fLJr(ia,ir,ith,iphi) = idealHist(ia, 2,ir,ith,iphi)
fLJs(ia,ir,ith,iphi) = idealHist(ia, 3,ir,ith,iphi)
fLJt(ia,ir,ith,iphi) = idealHist(ia, 4,ir,ith,iphi)
fCr(ia,ir,ith,iphi) = idealHist(ia, 8,ir,ith,iphi)
fCs(ia,ir,ith,iphi) = idealHist(ia, 9,ir,ith,iphi)
fCt(ia,ir,ith,iphi) = idealHist(ia,10,ir,ith,iphi)
end if
end do ! ir
end do ! iphi
end do ! ith

! Set all ln(g) values past the largest r bin to reach the cutoff to the
cutoff value.
igo = 0
do iphi = 1, histPhiBins
  do ith = 1, histCosThBins
    do ir = histDistBins, 1, -1
      if ((g(ia,ir,ith,iphi).lt.-abs(cut)).and.(igo.eq.0)) then
        g(ia,1:ir,ith,iphi) = -abs(cut)
        igo = 1
      end if
    end do
  end do
  igo = 0
end do
end do

! Set all LJ forces past (to the left of) the first (largest r) bin to reach the
cutoff to the cutoff value.
igo = 0; igo1 = 0; igo2 = 0
do iphi = 1, histPhiBins
  do ith = 1, histCosThBins
    do ir = histDistBins, 1, -1
      if ((fLJr(ia,ir,ith,iphi).gt.abs(cut)).and.(igo.eq.0)) then
        fLJr(ia,1:ir,ith,iphi) = abs(cut)
        igo = 1
      end if
      if ((fLJs(ia,ir,ith,iphi).gt.abs(cut)).and.(igo1.eq.0)) then

```

```

        fLJs(ia,1:ir,ith,iphi) = abs(cut)
        igo1 = 1
    end if
    if ((fLJt(ia,ir,ith,iphi).gt.abs(cut)).and.(igo2.eq.0)) then
        fLJt(ia,1:ir,ith,iphi) = abs(cut)
        igo2 = 1
    end if
end do
igo = 0; igo1 = 0; igo2 = 0
end do
end do

! Set all C forces past (to the left of) the first (largest r) bin to reach the
cutoff to the cutoff value.
igo = 0; igo1 = 0; igo2 = 0
do iphi = 1, histPhiBins
    do ith = 1, histCosThBins
        do ir = histDistBins, 1, -1
            if ((abs(fCr(ia,ir,ith,iphi)).gt.abs(cut)).and.(igo.eq.0)) then
                fCr(ia,1:ir,ith,iphi) = sign(real(1,dp),idealHist(ia,8,ir,ith,iphi))
* abs(cut)
                igo = 1
            end if
            if ((abs(fCs(ia,ir,ith,iphi)).gt.abs(cut)).and.(igo1.eq.0)) then
                fCs(ia,1:ir,ith,iphi) = sign(real(1,dp),idealHist(ia,9,ir,ith,iphi))
* abs(cut)
                igo1 = 1
            end if
            if ((abs(fCt(ia,ir,ith,iphi)).gt.abs(cut)).and.(igo2.eq.0)) then
                fCt(ia,1:ir,ith,iphi) = sign(real(1,dp),idealHist(ia,10,ir,ith,iphi))
) * abs(cut)
                igo2 = 1
            end if
        end do
        igo = 0; igo1 = 0; igo2 = 0
    end do
end do
! do ir = 1, histDistBins !debug
!   print*, 'g3D: ', ir, g(ia,ir,1,1) !debug
! end do !debug

! Average the 3D input histograms into 2D
write(*,*) 'Averaging input histograms from 3D into 2D for solute: ',ia

! Find the minimum value of u(phi; r,th) ==> u02D(r,th)
! dim=3 in this case means the phi dimension. Replace the array in phi at each r
,th with the minimum value of the array,
! making an array u02D(r,th).
u02D = minval(-g(ia,:::,dim=3)
do ir = 1, histDistBins !debug
!   print*, 'u0_2D: ',ir, u02D(ir,1) !debug
! end do !debug

do ith = 1, histCosThBins
    do ir = 1, histDistBins
        boltz_sum = 0_dp
        do iphi = 1, histPhiBins
            boltz = exp(g(ia,ir,ith,iphi) + u02D(ir,ith))
            !print*, 'boltz: ', g(ia,ir,ith,iphi), u02D(ir,ith), boltz !debug
            g2D(ia,ir,ith) = g2D(ia,ir,ith) + exp(g(ia,ir,ith,iphi)) ! g is g
            !print*, 'g2Dloop: ', ith,ir,iphi, g2D(ia,ir,ith) !debug
            fLJr2D(ia,ir,ith) = fLJr2D(ia,ir,ith) + (boltz * fLJr(ia,ir,ith,iphi))
! fLJ.r
            fLJs2D(ia,ir,ith) = fLJs2D(ia,ir,ith) + (boltz * fLJs(ia,ir,ith,iphi))
! fLJ.s

```



```

        fCr2D(ia,ir,ith) = fCr2D(ia,ir,ith) + (boltz * fCr(ia,ir,ith,iphi)) !
fC.r
        fCs2D(ia,ir,ith) = fCs2D(ia,ir,ith) + (boltz * fCs(ia,ir,ith,iphi)) !
fC.s
        boltz_sum = boltz_sum + boltz ! denominator for averaging over phi
    end do
    !print*, 'boltz_sum: ', boltz_sum !debug
    g2D(ia,ir,ith) = log(g2D(ia,ir,ith) / real(histPhiBins,dp)) ! finish
average over phi by dividing and converting to log(g)
    if (g2D(ia,ir,ith).lt.-abs(cut)) g2D(ia,ir,ith) = -abs(cut)
    fLJr2D(ia,ir,ith) = fLJr2D(ia,ir,ith) / boltz_sum
    fLJs2D(ia,ir,ith) = fLJs2D(ia,ir,ith) / boltz_sum
    fCr2D(ia,ir,ith) = fCr2D(ia,ir,ith) / boltz_sum
    fCs2D(ia,ir,ith) = fCs2D(ia,ir,ith) / boltz_sum
    end do
    !print*, ith, fLJr2D(ia,1:40,ith) !debug
end do
!
do ir = 1, histDistBins !debug
!
    print*, 'g2D: ', ir, g2D(ia,ir,1) !debug
!
end do !debug

! Average the 2D input histograms into 1D
write(*,*) 'Averaging input histograms from 2D into 1D for solute: ',ia

u01D(:) = minval(-g2D(ia, :, :), dim=2)
!
do ir = 1, histDistBins !debug
!
    print*, 'u0_1D: ', ir, u01D(ir) !debug
!
end do !debug

do ir = 1, histDistBins
    boltz_sum = 0_dp
    do ith = 1, histCosThBins
        boltz = exp(g2D(ia,ir,ith) + u01D(ir))
        !print*, 'boltz: ', g2D(ia,ir,ith), u01D(ir), boltz !debug
        g1D(ia,ir) = g1D(ia,ir) + exp(g2D(ia,ir,ith)) ! g1D is g
        fLJr1D(ia,ir) = fLJr1D(ia,ir) + (boltz * fLJr2D(ia,ir,ith)) ! fLJ.r
        fCr1D(ia,ir) = fCr1D(ia,ir) + (boltz * fCr2D(ia,ir,ith)) ! fC.r
        boltz_sum = boltz_sum + boltz ! denominator for averaging over theta
    end do
    !print*, 'boltz_sum: ', boltz_sum !debug
    g1D(ia,ir) = log(g1D(ia,ir) / real(histCosThBins,dp)) ! finish average over
cosTh by dividing and converting to log(g)
    fLJr1D(ia,ir) = fLJr1D(ia,ir) / boltz_sum
    fCr1D(ia,ir) = fCr1D(ia,ir) / boltz_sum
    end do
    !print*, 'fLJ1D: ', fLJr1D(ia,1:40) !debug
! After the averaging is done enforce the cutoff
do ir = 1, histDistBins
    if (g1D(ia,ir).lt.cut) then
        g1D(ia,ir) = cut
        ispline1D = ir
    end if
    if (fLJr1D(ia,ir).gt.abs(cut)) then
        fLJr1D(ia,ir) = -cut
    end if
    if (abs(fCr1D(ia,ir)).gt.abs(cut)) then
        fCr1D(ia,ir) = sign(real(1,dp),idealHist1D(ia,4,ir))*cut
    end if
end do
! spline g1D for the solute
call spline(histDist,g1D(ia,:),ispline1D,histDistBins,(idealHist1D(ia,2,
ispline1D)+idealHist1D(ia,4,ispline1D)),real(0,dp), &
& g1D2(ia,:))
call spline(histDist,fLJr1D(ia,:),ispline1D,histDistBins,idealHist1D(ia,3,
ispline1D),real(0,dp), fLJr1D2(ia,:))

```

```

    call spline(histDist,fCr1D(ia,:),ispline1D,histDistBins,idealHist1D(ia,5,
    ispline1D),real(0,dp), fCr1D2(ia,:))
end do ! ia

! note: write out the effective input histogram after averaging/alterations.
write(*,*) 'Writing input histogram after averaging/alterations to "input_hist.out"
...
open(91,file='input_hist.out',status='replace')
write(91,*) '# 1. Distance'
write(91,*) '# 2. g+'
write(91,*) '# 3. fLJ.r+'
write(91,*) '# 4. fC.r+'
write(91,*) '# 5. g-'
write(91,*) '# 6. fLJ.r-'
write(91,*) '# 7. fC.r-'
do ir = 1, histDistBins
    write(91,*) histDist(ir), g1D(1,ir), fLJr1D(1,ir), fCr1D(1,ir), g1D(2,ir),
    fLJr1D(2,ir), fCr1D(2,ir)
end do
close(91)

! note: write out the ideal histogram after averaging/alterations.
write(*,*) 'Writing ideal histogram after averaging/alterations to "ideal_hist.out"
...
open(92,file='ideal_hist.out',status='replace')
write(92,*) '# 1. Distance'
write(92,*) '# 2. g+'
write(92,*) '# 3. fLJ.r+'
write(92,*) '# 4. fC.r+'
write(92,*) '# 5. g-'
write(92,*) '# 6. fLJ.r-'
write(92,*) '# 7. fC.r-'
do ir = 1, histDistBins
    write(92,*) histDist(ir), -idealHist1D(1,1,ir), idealHist1D(1,2,ir), idealHist1D
    (1,4,ir), -idealHist1D(2,1,ir), &
    & idealHist1D(2,2,ir), idealHist1D(2,4,ir)
end do
close(92)

!debug
! write(*,*) 'Writing ideal histogram after averaging/alterations to "ideal_3D.out"
...
! open(93,file='ideal_3D.out',status='replace')
! write(93,*) '# 1. Distance'
! write(93,*) '# 2. cosTheta'
! write(93,*) '# 3. Phi'
! write(93,*) '# 4. g+'
! write(93,*) '# 5. fLJ.r+'
! write(93,*) '# 6. fC.r+'
! write(93,*) '# 7. g-'
! write(93,*) '# 8. fLJ.r-'
! write(93,*) '# 9. fC.r-'
! do ir = 1, histDistBins
!     do ith = 1, histCosThBins
!         do iphi = 1, histPhiBins
!             write(93,*) histDist(ir), histCosTh(ith), histPhi(iphi), -idealHist(1,1,
! ir,ith,iphi), idealHist(1,2,ir,ith,iphi), &
!             & idealHist(1,8,ir,ith,iphi), -idealHist(2,1,ir,ith,iphi), idealHist
! (2,2,ir,ith,iphi), idealHist(2,8,ir,ith,iphi)
!         end do
!     end do
! end do
! close(93)

!debug    difference between ideal and measured

```

```

!rTmp = int(real(4,dp)/histDistStepSize)
!print*, rTmp, histDistStepSize
!do ith=1,histCosThBins
    !write(55,*) histCosTh(ith), g2D(rTmp,ith), idealhist2D(1,rTmp,ith)
!end do

!debug
!do ir=1,histdistbins
    !write(45,*) histDist(ir), g1D(ir), g1D2(ir), idealHist1D(1,ir), fLJr1D(ir),
    idealHist1D(2,ir)
!end do
!do ir=1,100*histdistbins
    !xx = ir*(histDistStepSize/100_dp)
    !call splint(histDist,g1D,g1D2,histDistBins,xx, yy)
    !write(55,*) xx, yy
!end do
!write(65,*) histDist(ispline1D), g1D(ispline1D)

end subroutine spline_hist_array

! read LJ--LJ displacements from file
subroutine R_list
    use cfgData; use ctrlData
    implicit none
    integer          :: ios, i
    character(len=16) :: junk
    character(len=128) :: line

    if (c_explicit_R .eq. 'no') then
        explicit_R = .false.
    else if (c_explicit_R .eq. 'yes') then
        explicit_R = .true.

        ios = 0; crdLines = -1
        open(20,file='crd_list.out',status='old')
        do while(ios>=0)
            read(20,'(a)',IOSTAT=ios) line
            crdLines = crdLines + 1
        end do
        close(20)

        allocate( explicitDist(crdLines) )

        ios = 0
        open(20,file='crd_list.out',status='old')
        do i = 1, crdLines
            read(20,*,iostat=ios) junk, explicitDist(i)
        end do
        close(20)
    end if
end subroutine R_list

! setup for the average force integral
subroutine setup_compute_avg_force
    use cfgData; use angleData; use ctrlData
    implicit none
    integer :: i
    real(kind=dp) :: psiLF

    write(*,*) "Setting up for average force iteration..."

    if (explicit_R .eqv. .true.) then

```

```

    cfgRBins = crdLines
    write(*,*) "Number of R Bins:      ", cfgRBins
else if (explicit_R .eqv. .false.) then
    cfgRBins = int( (R_max - R_min)/RStepSize + 1 )
    if (cfgRBins .eq. 0) then
        cfgRBins = 1
    end if
    write(*,*) "Number of R Bins:      ", cfgRBins
end if
xBins = int( (2 * xz_range)/xzStepSize )
write(*,*) "Number of X Bins:      ", xBins
zBins = int( (xz_range)/xzStepSize )
write(*,*) "Number of Z Bins:      ", zBins

! allocate array sizes for axes and average force
allocate( R_axis(cfgRBins), fAvg(2,cfgRBins), x_axis(xBins), z_axis(zBins) )
R_axis = 0_dp; fAvg = 0_dp; x_axis = 0_dp; z_axis = 0_dp

! allocate arrays for control arrays
! frcSPA(solutes, LJ/C, x, z)
allocate( frcSPA(2, 2, xBins, zBins), grSPA(xBins, zBins) )

! Distance Axes
do i = 1, cfgRBins
    if (explicit_R .eqv. .true.) then
        R_axis(i) = explicitDist(i)
    else if (explicit_r .eqv. .false.) then
        R_axis(i) = (i-1) * RStepSize + R_min
    end if
end do
do i = 1, xBins
    x_axis(i) = (i-1) * xzStepSize - xz_range + xzStepSize/2_dp
end do
do i = 1, zBins
    z_axis(i) = (i-1) * xzStepSize + xzStepSize/2_dp
end do

! ANGLES
allocate( cosThetaLF(cfgCosThBins), sinThetaLF(cfgCosThBins), sinPsiLF(cfgPsiBins),
        cosPsiLF(cfgPsiBins) )

! Theta
! tilt off of z
cfgCosThStepSize = (cosTh_max - cosTh_min) / real(cfgCosThBins, dp)
do i = 1, cfgCosThBins
    cosThetaLF(i) = (i-0.5_dp)*cfgCosThStepSize - cosTh_max
    sinThetaLF(i) = sqrt(abs(1_dp-cosThetaLF(i)**2))
end do
write(*,*) "Config Cos(Theta) Step Size:      ", cfgCosThStepSize

! Psi
! processison about z
cfgPsiStepSize = (psi_max - psi_min) / real(cfgPsiBins, dp)
do i = 1, cfgPsiBins
    psiLF = (i+0.5_dp)*cfgPsiStepSize
    sinPsiLF(i) = sin(psiLF)
    cosPsiLF(i) = cos(psiLF)
end do
write(*,*) "Config Psi Step Size:      ", cfgPsiStepSize
end subroutine setup_compute_avg_force

! do the average force integral
subroutine compute_avg_force

```

```

use cfgData; use histData; use angleData; use ctrlData; use constants; use
functions
implicit none
integer :: r, i, j, ip, omp_get_thread_num !, omp_get_num_threads
real(kind=dp) :: gx1, gx2, f1lj, f1c, f2lj, f2c, f

write(*,*) "Computing average force..."
flush(6)

! Calculate the average force integral for top half of bisecting plane of cylinder
do r = 1, cfgRBins ! loop lj--lj distances
  frcSPA = 0_dp; grSPA = 0_dp
  !$omp PARALLEL DEFAULT( none ) &
  !$omp PRIVATE( ip, i, j, gx1, gx2, f1lj, f1c, f2lj, f2c, f ) &
  !$omp SHARED( nThreads, r, xBins, zBins, cut, R_axis, x_axis, z_axis, histDist,
histDistBins, g1D, g1D2, fLJr1D, fLJr1D2, &
!$omp& fCr1D, fCr1D2, frcSPA, grSPA ) &
  !$omp NUM_THREADS( nThreads )
  if ((omp_get_thread_num().eq.0).and.(r.eq.1)) then
    write(*,*) 'Parallel CPUs:          ', nThreads
    !write(*,*) 'Parallel CPUs:          ', omp_get_num_threads()
    flush(6)
  end if
  !$omp DO SCHEDULE( guided )
  do ip = 1, (xBins*zBins)
    ! Convert single index 'ip' to the x and z indices 'i' and 'j' respectively.
    i = int((ip-1)/zBins)+1 ! x integer
    j = mod(ip-1,zBins)+1 ! z integer

    rSolv1(1) = -R_axis(r)/2_dp - x_axis(i)
    rSolv1(2) = 0_dp
    rSolv1(3) = -z_axis(j)
    rSolv1n(1) = euclid_norm(rSolv1)
    rSolv2(1) = R_axis(r)/2_dp - x_axis(i)
    rSolv2(2) = 0_dp
    rSolv2(3) = -z_axis(j)
    rSolv2n(1) = euclid_norm(rSolv2)

    if (rSolv1n(1).gt.histDist(histDistBins)) then
      gx1 = 0_dp
    else
      call splint(histDist,g1D(1,:),g1D2(1,:),histDistBins,rSolv1n(1), gx1) !
solute 1
    end if
    if (rSolv2n(1).gt.histDist(histDistBins)) then
      gx2 = 0_dp
    else
      call splint(histDist,g1D(2,:),g1D2(2,:),histDistBins,rSolv2n(2), gx2) !
solute 2
    end if
    gx1 = exp(gx1+gx2)

    if (gx1 .gt. 1d-6) then ! if gx1 == 0 then don't waste time with the rest of
the calculation
      if (rSolv1n(1).gt.histDist(histDistBins)) then ! solute 1
        f1lj = 0_dp; f1c = 0_dp
      else
        call splint(histDist,fLJr1D(1,:),fLJr1D2(1,:),histDistBins,rSolv1n(1), f
)
        f1lj = f
        call splint(histDist,fCr1D(1,:),fCr1D2(1,:),histDistBins,rSolv1n(1), f)
        f1c = f
      end if
      if (rSolv2n(1).gt.histDist(histDistBins)) then ! solute 2
        f2lj = 0_dp; f2c = 0_dp

```

```

        else
            call splint(histDist, fLJr1D(2,:), fLJr1D2(2,:), histDistBins, rSolvN(2), f
)
            f2lj = f
            call splint(histDist, fCr1D(2,:), fCr1D2(2,:), histDistBins, rSolvN(2), f)
            f2c = f
        end if

        frcSPA(1,1,i,j) = frcSPA(1,1,i,j) + (gx1 * f1lj * (-rSolv1(1)/rSolvN(1)))
! (f.r)*g.R^{hat} solute 1 lj
        frcSPA(1,2,i,j) = frcSPA(1,2,i,j) + (gx1 * f1c * (-rSolv1(1)/rSolvN(1)))
! (f.r)*g.R^{hat} solute 1 c
        frcSPA(2,1,i,j) = frcSPA(2,1,i,j) + (gx1 * f2lj * ( rSolv2(1)/rSolvN(2)))
! (f.r)*g.R^{hat} solute 2 lj
        frcSPA(2,2,i,j) = frcSPA(2,2,i,j) + (gx1 * f2c * ( rSolv2(1)/rSolvN(2)))
! (f.r)*g.R^{hat} solute 2 c
        grSPA(i,j) = grSPA(i,j) + gx1 ! gx1 is the SPA at this point
    end if
end do !ip
!$omp END DO
!$omp END PARALLEL

! Add each cell forces to average and normalize
do i = 1, xBins
    do j = 1, zBins
        fAvg(1,r) = fAvg(1,r) + (frcSPA(1,1,i,j) + frcSPA(2,1,i,j)) * real(0.5,dp)
* z_axis(j) ! lj
        fAvg(2,r) = fAvg(2,r) + (frcSPA(1,2,i,j) + frcSPA(2,2,i,j)) * real(0.5,dp)
* z_axis(j) ! Coulomb
        !fAvg(1,r) = fAvg(1,r) + (frcSPA(1,1,i,j) + frcSPA(2,1,i,j)) * z_axis(j) !
lj
        !fAvg(2,r) = fAvg(2,r) + (frcSPA(1,2,i,j) + frcSPA(2,2,i,j)) * z_axis(j) !
c
        frcSPA(1,1,i,j) = frcSPA(1,1,i,j)/grSPA(i,j)
        frcSPA(1,2,i,j) = frcSPA(1,2,i,j)/grSPA(i,j)
        frcSPA(2,1,i,j) = frcSPA(2,1,i,j)/grSPA(i,j)
        frcSPA(2,2,i,j) = frcSPA(2,2,i,j)/grSPA(i,j)
    end do !z again
end do !x again
call write_test_out(r) ! write grSPA and frcSPA arrays

! NOTE : After the fact multiply all elements by 2*pi*density/8/pi/pi (2*2pi*pi
/3 (4pi**2)/3 steradians from orientations)
! Number density of chloroform per Angstrom**3 == 0.00750924
fAvg(1,r) = fAvg(1,r)*2*pi*density*xzStepSize*xzStepSize
fAvg(2,r) = fAvg(2,r)*2*pi*density*xzStepSize*xzStepSize
end do !r

end subroutine compute_avg_force

! integrate the average force from 'compute_avg_force' to get the PMF.
subroutine integrate_force
    use cfgData; use ctrlData
    implicit none
    integer :: d, f

    allocate( u_dir(2, cfgRBins) )
    u_dir = 0_dp

    do f = 1, 2
        if (explicit_R .eqv. .false.) then
            do d = 1, cfgRBins
                if (d .eq. 1) then
                    u_dir(f, cfgRBins) = fAvg(f, cfgRBins) * RStepSize
                end if
            end do
        end if
    end do
end subroutine integrate_force

```

```

        else
            u_dir(f,cfgRBins-(d-1)) = u_dir(f,cfgRBins-(d-2)) + fAvg(f,cfgRBins-(d
-1)) * RStepSize
        end if
    end do
    else if (explicit_R .eqv. .true.) then
        do d = 1, cfgRBins
            if (d .eq. 1) then
                u_dir(f,cfgRBins) = fAvg(f,cfgRBins) * (R_axis(cfgRBins)-R_axis(
cfgRBins-1))
                !print*, (R_axis(cfgRBins)-R_axis(cfgRBins-1))
            else
                ! FIXME: is the delta R part of this correct?
                u_dir(f,cfgRBins-(d-1)) = u_dir(f,cfgRBins-(d-2)) + fAvg(f,cfgRBins-(d
-1)) * &
                    (R_axis(cfgRBins-(d-1))-R_axis(cfgRBins-d))
                ! it looks like the first value is getting printed twice. Also, the
values might be wrong. Should it be (d-1) and
                ! (d-0)? instead of -2 and -1?
                !print*, (R_axis(cfgRBins-(d-1))-R_axis(cfgRBins-d))
            end if
        end do
    end if
end do
end subroutine integrate_force

! write force out and g(r) out to compare against explicit
subroutine write_test_out(r)
    use cfgData; use ctrlData
    implicit none
    integer :: r, i_f, i, j
    character(len=32) :: temp, filename
    character(len=8) :: frmt

    i_f = (r-1) * int(RStepSize*10)
    frmt = '(I3.3)' ! an integer of width 3 with zeroes on the left
    write(temp,frmt) i_f ! converting integer to string using 'internal file'
    filename='hist1D_output.'//trim(temp)//'.dat'

    open(35,file=filename,status='replace')
    write(6,*) "Writing test file: ", filename
    write(35,*) "# 1. X Distance"
    write(35,*) "# 2. Z Distance"
    write(35,*) "# 3. g(r)"
    write(35,*) "# 4. Force.r + LJ"
    write(35,*) "# 5. Force.r + Coulomb"
    write(35,*) "# 6. Force.r - LJ"
    write(35,*) "# 7. Force.r - Coulomb"
    write(35,*) "# "
    do j = 1, zBins
        do i = 1, xBins
            write(35,898) x_axis(i), z_axis(j), grSPA(i,j), frcSPA(1,1,i,j), frcSPA(1,2,i
,j), frcSPA(2,1,i,j), frcSPA(2,2,i,j)
        end do
    end do
    close(35)

    flush(6)

898     format (2(1x,es14.7),5(1x,es14.7))
end subroutine write_test_out

```





```

real(kind=dp) :: RStepSize, xzStepSize, R_min, R_max, xz_range, cfgCosThStepSize,
  cfgPsiStepSize, T, cut, offset
character(len=8) :: c_explicit_R
integer :: cfgRBins, cfgCosThBins, cfgPhiBins, cfgPsiBins, radius
!
integer :: xBins, zBins
real(kind=dp) :: density = 0.00750924_dp ! numerical density of chloroforms per
  Angstrom**3
real(kind=dp) :: cosTh_max = 1_dp
real(kind=dp) :: cosTh_min = -1_dp
real(kind=dp) :: phi_max = 2_dp*pi/3_dp ! 2pi/3 is sufficient for a molecule with
  C3 symmetry.
real(kind=dp) :: phi_hmax = pi/3_dp
real(kind=dp) :: phi_min = 0_dp
real(kind=dp) :: psi_max = 2_dp*pi
real(kind=dp) :: psi_min = 0_dp

end module cfgData

! data for calculating cosTh value.
module angleData
  use prec
  real(kind=dp), allocatable :: sinThetaLF(:), cosThetaLF(:), sinPhiLF(:), cosPhiLF(:)
  , sinPsiLF(:), cosPsiLF(:)
  real(kind=dp) :: rSolv1(3), rSolv2(3), rSolvn(2), cosTh(2), phi(2), sSolv1(3),
  tSolv1(3), sSolvln, tSolvln

  !$omp THREADPRIVATE( rSolv1, rSolv2, rSolvn, sSolv1, sSolvln, tSolv1, tSolvln,
  cosTh, phi )

end module angleData

! testing arrays for force and g(r)
module ctrlData
  use prec
  real(kind=dp), allocatable :: frcSPA(:, :, :), grSPA(:, :, :), explicitDist(:)
  integer :: crdLines
  logical :: explicit_R

end module ctrlData

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

program compute_avgForce
  use prec
  implicit none
  character(len=64) :: histFile, cfgFile, outFile
  real(kind=dp) :: omp_get_wtime, ti, tf, seconds
  integer :: hours, minutes

  ti = omp_get_wtime()

  ! make list of average direct force from 'collapsed' file.
  call parse_command_line(cfgFile)

  ! read config file
  call read_cfg(cfgFile, histFile, outFile)

  ! make list of average direct force from 'collapsed' file.
  call make_hist_table(histFile)

  ! Now that we have the relevant information spline the g and f arrays along r.

```

```

call spline_hist_array

! read in LJ--LJ dist array from file
call R_list

! setup for computing the average force integral.
call setup_compute_avg_force

! compute average force integral.
call compute_avg_force

! integrate average force to get PMF.
call integrate_force

! write PMF output file
call write_output(outFile)

! Write time taken to finish calculation.
tf = omp_get_wtime()

hours = (tf-ti)/3600
minutes = mod((tf-ti),3600d0)/60
seconds = mod(mod((tf-ti),3600d0),60d0)

write(*,*) "~~~~~"
write(*,'(a,i4,a,i2,a,f6.3,a)') "Total time elapsed: ", hours, "h ", minutes, "m
", seconds, "s"

flush(6)

end program compute_avgForce

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!! Subroutines !!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

! parse commandline for relevant files.
subroutine parse_command_line(cfgFile)
  implicit none
  character(len=64) :: cfgFile
  character(len=16) :: arg
  integer :: i
  logical :: cfgFileFlag, cfgExist

  cfgFileFlag = .false.
  cfgExist = .false.
  i=1
  do
    call get_command_argument(i,arg)
    select case (arg)

    case ('-cfg')
      i = i+1
      call get_command_argument(i,cfgFile)
      cfgFileFlag=.true.
      INQUIRE(FILE=cfgFile, EXIST=cfgExist)
      write(*,*) 'Config File: ', cfgFile
      write(*,*) 'Config File Exists: ', cfgExist
    case default
      write(*,*) 'Unrecognized command-line option: ', arg
      write(*,*) 'Usage: compute_avgForce.x -cfg [cfg file]'
      stop
    end select
  end do
end subroutine parse_command_line

```

```

        i = i+1
        if (i.ge.command_argument_count()) exit
    end do

    if (cfgFileFlag.eqv..false.) then
        write(*,*) "Must provide a cfg file using command line argument -cfg [cfg file
        name]"
        stop
    end if

    ! 'ERROR STOP' if either file doesn't exist
    if (cfgExist.eqv..false.) then
        write(*,*) 'cfg file does not exist'
        error stop
    end if

    flush(6)

end subroutine parse_command_line

! read python cfg file for g(r) parameters
subroutine read_cfg(cfgFile, histFile, outFile)
    use cfgData
    implicit none
    character(len=64) :: cfgFile, histFile, outFile
    character(len=128) :: line
    character(len=32) :: firstWord, sep
    integer :: ios
    logical :: outFileFlag, histFileFlag, histExist, RstepSizeFlag, xzStepSizeFlag,
    RmaxFlag, RminFlag, xzRangeFlag, &
    & thetaBinsFlag, phiBinsFlag, psiBinsFlag, c_explicit_RFlag, TFlag, cutFlag,
    radiusFlag, offsetFlag

    histFileFlag = .false.
    histExist = .false.
    outFileFlag = .false.
    RstepSizeFlag = .false.
    xzStepSizeFlag = .false.
    c_explicit_RFlag = .false.
    RmaxFlag = .false.
    RminFlag = .false.
    xzRangeFlag = .false.
    thetaBinsFlag = .false.
    phiBinsFlag = .false.
    psiBinsFlag = .false.
    TFlag = .false.
    cutFlag = .false.
    radiusFlag = .false.
    offsetFlag = .false.

    ios = 0

    open(20,file=cfgFile)
    do while(ios>=0)
        read(20,'(a)',IOSTAT=ios) line
        call split(line,'=',firstWord, sep)
        if (line .ne. "") then
            if (firstWord .eq. "hist_file") then
                read(line,'(a)') histFile
                write(*,*) "Histogram File:           ", histFile
                histFileFlag = .true.
                INQUIRE(FILE=histFile, EXIST=histExist) ! check if it exists
            else if (firstWord .eq. "out_file") then
                read(line,*) outFile
            end if
        end if
    end do
end subroutine read_cfg

```

```

        write(*,*) "Output File:           ", outFile
        outFileFlag = .true.
    else if (firstWord .eq. "RStepSize") then
        read(line,*) RStepSize
        write(*,*) "PMF Step Size:           ", RStepSize
        RstepSizeFlag = .true.
    else if (firstWord .eq. "xzStepSize") then
        read(line,*) xzStepSize
        write(*,*) "Solvent Grid Step Size:           ", xzStepSize
        xzStepSizeFlag = .true.
    else if (firstWord .eq. "explicit_R") then
        read(line,*) c_explicit_R
        write(*,*) "Use Explicit R Values:           ", c_explicit_R
        c_explicit_RFlag = .true.
    else if (firstWord .eq. "R_max") then
        read(line,*) R_max
        write(*,*) "R Maximum Value:           ", R_max
        RmaxFlag = .true.
    else if (firstWord .eq. "R_min") then
        read(line,*) R_min
        write(*,*) "R Minimum Value:           ", R_min
        RminFlag = .true.
    else if (firstWord .eq. "xz_range") then
        read(line,*) xz_range
        write(*,*) "XZ - Range:           ", xz_range
        xzRangeFlag = .true.
    else if (firstWord .eq. "theta_bins") then
        read(line,*) cfgCosThBins
        write(*,*) "Theta Bins:           ", cfgCosThBins
        thetaBinsFlag= .true.
    else if (firstWord .eq. "phi_bins") then
        read(line,*) cfgPhiBins
        write(*,*) "Phi Bins:           ", cfgPhiBins
        phiBinsFlag= .true.
    else if (firstWord .eq. "psi_bins") then
        read(line,*) cfgPsiBins
        write(*,*) "Psi Bins:           ", cfgPsiBins
        psiBinsFlag= .true.
    else if (firstWord .eq. "temperature") then
        read(line,*) T
        write(*,*) "Temperature (K):           ", T
        TFlag= .true.
    else if (firstWord .eq. "bicubic_cutoff") then
        read(line,*) cut
        write(*,*) "Bicubic/Bilinear Cutoff:           ", cut
        cutFlag= .true.
    else if (firstWord .eq. "solute_radius") then
        read(line,*) radius
        write(*,*) "Solute radius:           ", radius
        radiusFlag= .true.
    else if (firstWord .eq. "offset") then
        read(line,*) offset
        write(*,*) "Solvent offset distance:           ", offset
        offsetFlag= .true.
    end if
end if
end do
close(20)

if (histFileFlag.eqv..false.) then
    write(*,*) "Config file must have a 'hist_file' value"
    stop
end if
if (histExist.eqv..false.) then

```

```

    write(*,*) "Config file must point to a 'hist_file' that exists: ", histFile, "
    doesn't exist."
    stop
end if
if (outFileFlag.eqv..false.) then
    write(*,*) "Config file must have a 'out_file' value"
    stop
end if
if (RstepSizeFlag.eqv..false.) then
    write(*,*) "Config file must have a 'RStepSize' value"
    stop
end if
if (xzStepSizeFlag.eqv..false.) then
    write(*,*) "Config file must have a 'xzStepSize' value"
    stop
end if
if (outFileFlag.eqv..false.) then
    write(*,*) "Config file must have a 'out_file' value"
    stop
end if
if (c_explicit_RFlag.eqv..false.) then
    write(*,*) "Config file must have a 'explicit_R' value"
    stop
end if
if (RmaxFlag.eqv..false.) then
    write(*,*) "Config file must have a 'R_max' value"
    stop
end if
if (RminFlag.eqv..false.) then
    write(*,*) "Config file must have a 'R_min' value"
    stop
end if
if (xzRangeFlag.eqv..false.) then
    write(*,*) "Config file must have a 'xz_range' value"
    stop
end if
if (thetaBinsFlag.eqv..false.) then
    write(*,*) "Config file must have a 'theta_bins' value"
    stop
end if
if (phiBinsFlag.eqv..false.) then
    write(*,*) "Config file must have a 'phi_bins' value"
    stop
end if
if (psiBinsFlag.eqv..false.) then
    write(*,*) "Config file must have a 'psi_bins' value"
    stop
end if
if (TFlag.eqv..false.) then
    write(*,*) "Config file must have a 'temperature' value"
    stop
end if
if (cutFlag.eqv..false.) then
    write(*,*) "Config file must have a 'bicubic_cutoff' value"
    stop
end if
if (radiusFlag.eqv..false.) then
    write(*,*) "Config file must have a 'solute_radius' value"
    stop
end if
if (offsetFlag.eqv..false.) then
    write(*,*) "Config file must have a 'offset' value"
    stop
end if

```

```

flush(6)

end subroutine read_cfg

! read force file and make a lookup table.
subroutine make_hist_table(histFile)
  use histData
  use cfgData
  implicit none
  character(len=64) :: histFile
  character(len=64) :: junk
  character(len=512) :: line
  integer :: ios, ios2, i, j, k, nHistLines
  real(kind=dp),allocatable :: histTmp(:, :)

  ! read number of lines in histFile and allocate that many points in temporary
  histogram list, histTmp.
  ios = 0; nHistLines = -1
  open(20,file=histFile)
  do while(ios>=0)
    read(20,'(a)',IOSTAT=ios) line
    if (line(1:1) .ne. "#") then
      nHistLines = nHistLines + 1
    end if
  end do
  close(20)
  !write(*,*) "nHistLines", nHistLines

  allocate( histTmp(8,nHistLines) )

  ! populate hist arrays
  ios = 0; i = 1
  open(20,file=histFile)
  ! read file ignoring comment lines at the beginning
  do while(ios>=0)
    read(20,'(a)',IOSTAT=ios) line
    if ((line(1:1) .ne. "#") .and. (ios .ge. 0)) then
      !
      ! (r)-          <f.r>+          dist          <f.s>+          cos(Th)          phi/3          g(r)+          g
      !          <f.t>+
      read(line,*) histTmp(1,i), histTmp(2,i), histTmp(3,i), histTmp(4,i), junk,
      histTmp(5,i), histTmp(6,i), histTmp(7,i), &
      !          gc(r)+          gc(r)-
      & histTmp(8,i), junk
      i = i + 1
    end if
  end do
  close(20)

  ! Unique value determination
  do i = 1, nHistLines
    if (i .eq. 1) then
      histDistBins = 1
      ios = 0; ios2 = 0
    else ! i = 2, nHistLines
      if (( histTmp(1,i) .lt. (histTmp(1,i-1)-1d-6) ) .or. ( histTmp(1,i) .gt. (
      histTmp(1,i-1)+1d-6) )) then
        ! note: this statement will trigger when a value in the first column (dist
        ) is different than the value in the row
        ! before it.
        histDistBins = histDistBins + 1
      end if
      if (( histTmp(2,i) .gt. (histTmp(2,i)-1d-6) ) .and. ( histTmp(2,i) .lt. (
      histTmp(2,i)+1d-6) ) .and. ( ios .eq. 0 ) .and. &
      & ( ios2 .eq. 1 )) then

```

```

! note: this statement will trigger when i = histCosThBins+1 because it
finds the first repeated element
    histCosThBins = (i - 1)/histPhiBins
    ios = 1
end if
    if (( histTmp(3,i) .gt. (histTmp(3,1)-1d-6) ) .and. ( histTmp(3,i) .lt. (
histTmp(3,1)+1d-6) ) .and. ( ios2 .eq. 0 )) then
! note: this statement will trigger when i = histPhiBins+1 because it
finds the first repeated element
    histPhiBins = i - 1
    ios2 = 1
end if
end if
end do
write(*,*) "Histogram Distance Bins: ", histDistBins
write(*,*) "Histogram Cosine Theta Bins: ", histCosThBins
write(*,*) "Histogram Phi Bins: ", histPhiBins

allocate( histDist(histDistBins), histCosTh(histCosThBins), histPhi(histPhiBins), g
(histDistBins,histCosThBins,histPhiBins), &
& fr(histDistBins,histCosThBins,histPhiBins), fs(histDistBins,histCosThBins,
histPhiBins), &
& ft(histDistBins,histCosThBins,histPhiBins), gc(histDistBins,histCosThBins,
histPhiBins) )

! populate arrays that will be used in the rest of the calculation from temp array
do i = 1, histDistBins ! the values written out from python script are at half
-bin distances
    histDist(i) = histTmp(1,histCosThBins*histPhiBins*(i-1)+1)
end do
do i = 1, histCosThBins
    histCosTh(i) = histTmp(2,histPhiBins*(i-1)+1)
end do
do i = 1, histPhiBins
    histPhi(i) = histTmp(3,i)
end do

do i = 1, histDistBins
    do j = 1, histCosThBins
        do k = 1, histPhiBins
            g(i,j,k) = histTmp(4, (i-1)*histCosThBins*histPhiBins + (j-1)*histPhiBins
+ k) ! g(r,cos,phi) currently g
            fr(i,j,k) = histTmp(5, (i-1)*histCosThBins*histPhiBins + (j-1)*histPhiBins
+ k) ! <f.r>(r,cos,phi)
            fs(i,j,k) = histTmp(6, (i-1)*histCosThBins*histPhiBins + (j-1)*histPhiBins
+ k) ! <f.s>(r,cos,phi)
            ft(i,j,k) = histTmp(7, (i-1)*histCosThBins*histPhiBins + (j-1)*histPhiBins
+ k) ! <f.t>(r,cos,phi)
            gc(i,j,k) = histTmp(8, (i-1)*histCosThBins*histPhiBins + (j-1)*histPhiBins
+ k) ! gc(r,cos,phi)
        end do
    end do
end do

histDistStepSize = histDist(2) - histDist(1)
write(*,*) "Histogram Distance Step Size: ", histDistStepSize
histCosThStepSize = histCosTh(2) - histCosTh(1)
write(*,*) "Histogram Cosine Theta Step Size: ", histCosThStepSize
histPhiStepSize = histPhi(2) - histPhi(1)
write(*,*) "Histogram Phi Step Size: ", histPhiStepSize

flush(6)

end subroutine make_hist_table

```

```

! spline the r dimension of each theta phi stack and then average over phi for 2D
subroutine spline_hist_array
  use constants
  use functions
  use histData
  use cfgData
  use idealSolv
  implicit none
  integer                :: i, ir, ith, iphi, imin, igo, ir2, igo1, igo2
  real(kind=dp)          :: x, y, norm_factor, boltz, boltz_sum
  real(kind=dp),allocatable :: idealHist(:,:,:), idealHist2D(:,:,:), u0(:,:)
  integer,allocatable    :: ispline(:,:), ispline2D(:)
  !real(kind=dp)        :: xx, yy !debug

  write(*,*) 'Editing input histogram arrays with ideal arrays in 3D...'

  ! Calculate a 4D array idealHist(g/f,r,th,phi)
  allocate( idealHist(7,histDistBins,histCosThBins,histPhiBins), ispline(
    histCosThBins,histPhiBins), ispline2D(histCosThBins) )
  idealHist = 0_dp; ispline = 0_dp
  call ideal_CL3(histDistBins,histDistStepSize,histCosThBins,cosTh_min,cosTh_max,
    histPhiBins,phi_min,phi_hmax,radius,offset,T, &
    & idealHist)

  ! Edit the input hist arrays to more smoothly transition to -/+ infinity with the
  help of idealHist.
  do ith = 1, histCosThBins
    do iphi = 1, histPhiBins
      imin = 0
      ! Normalization factor or each theta phi array.
      norm_factor = gc(histDistBins,ith,iphi)/(g(histDistBins,ith,iphi)*4*pi*
        histDist(histDistBins)**2)
      ! Find the first non-zero g(r) bin for each theta/phi array and set 'imin' to
      that 'ir' index
find:      do ir = 1, histDistBins
            if (g(ir,ith,iphi).gt.1d-6) then
              imin = ir
              exit find
            end if
          end do find

      ! Note: Add the ideal values to bins with no sampling. And half counts to
      bins that probably should have had sampling.
      igo = 0
      do ir = histDistBins, 1, -1
        if (ir.ge.imin) then
          if (g(ir,ith,iphi).gt.1d-6) then
            g(ir,ith,iphi) = log(g(ir,ith,iphi))
          else ! note: This is a zero bin where there probably should have been
            something. So put a half count in.
            g(ir,ith,iphi) = log(real(0.5,dp)/(4*pi*histDist(ir)**2)/norm_factor
          )

          fr(ir,ith,iphi) = idealHist(2,ir,ith,iphi)
          fs(ir,ith,iphi) = idealHist(3,ir,ith,iphi)
          ft(ir,ith,iphi) = idealHist(4,ir,ith,iphi)
        end if
        else if (ir.lt.imin) then ! .lt.imin ==> in the region of no sampling. Set
        the FE (log(g)) to the direct energy shifted by
        ! a constant energy term, which is the last sampled indirect energy.
        ! ln(g(r<r0)) = -u_dir(r)/T - ( u_pmf(r0)/T - u_dir(r0)/T )
        ! ln(g(r<r0)) = -u_dir(r)/T + ln(g(r0)) - u_dir(r0)/T
          g(ir,ith,iphi) = ( -idealHist(1,ir,ith,iphi) + idealHist(1,imin,ith,
            iphi) ) + g(imin,ith,iphi)

```



```

        if ((g(ir,ith,iphi).lt.cut).and.(igo.eq.0)) then ! the largest r to go
past the cutoff
            ispline(ith,iphi) = ir
            igo = 1
        end if
        fr(ir,ith,iphi) = idealHist(2,ir,ith,iphi)
        fs(ir,ith,iphi) = idealHist(3,ir,ith,iphi)
        ft(ir,ith,iphi) = idealHist(4,ir,ith,iphi)
    end if
end do
end do
end do

! Set all forces past the first (largest r) bin to reach the cutoff to the cutoff
value. !debug
! igo = 0; igo1 = 0; igo2 = 0
! do iphi = 1, histPhiBins
!     do ith = 1, histCosThBins
!         do ir = histDistBins, 1, -1
!             if ((fr(ir,ith,iphi).gt.-cut).and.(igo.eq.0)) then
!                 fr(1:ir,ith,iphi) = -cut
!                 igo = 1
!             end if
!             if ((fs(ir,ith,iphi).gt.-cut).and.(igo1.eq.0)) then
!                 fs(1:ir,ith,iphi) = -cut
!                 igo1 = 1
!             end if
!             if ((ft(ir,ith,iphi).gt.-cut).and.(igo2.eq.0)) then
!                 ft(1:ir,ith,iphi) = -cut
!                 igo2 = 1
!             end if
!         end do
!         igo = 0; igo1 = 0; igo2 = 0
!     end do
! end do

! Average the 3D input histograms into 2D
write(*,*) 'Averaging input histograms from 3D into 2D...'

allocate( g2D(histDistBins,histCosThBins), fr2D(histDistBins,histCosThBins), fs2D(
histDistBins,histCosThBins), &
& u0(histDistBins,histCosThBins) )
g2D = 0_dp; fr2D = 0_dp; fs2D = 0_dp; u0 = 0_dp

! Find the minimum value of u(phi; r,th) ==> u0(r,th)
! dim=3 in this case means the phi dimension. Replace the array in phi at each r,th
with the minimum value of the array, making
! an array u0(r,th).
u0 = minval(-g(:, :, :), dim=3)

do ith = 1, histCosThBins
do ir = 1, histDistBins
boltz_sum = 0_dp
do iphi = 1, histPhiBins
boltz = exp(g(ir,ith,iphi) + u0(ir,ith))
g2D(ir,ith) = g2D(ir,ith) + exp(g(ir,ith,iphi)) ! g is g
fr2D(ir,ith) = fr2D(ir,ith) + (boltz * fr(ir,ith,iphi)) ! f.r
fs2D(ir,ith) = fs2D(ir,ith) + (boltz * fs(ir,ith,iphi)) ! f.s
boltz_sum = boltz_sum + boltz ! denominator for averaging over phi
end do
g2D(ir,ith) = log(g2D(ir,ith) / real(histPhiBins,dp)) ! finish average over
phi by dividing and convert to log(g)
fr2D(ir,ith) = fr2D(ir,ith) / boltz_sum
fs2D(ir,ith) = fs2D(ir,ith) / boltz_sum
end do
end do
end do

```

```

! After the average is done enforce lowest value to cutoff
do ir2 = 1, histDistBins
  if (g2D(ir2,ith).lt.cut) then
    g2D(ir2,ith) = cut
    fr2D(ir2,ith) = -cut
    fs2D(ir2,ith) = -cut
    ispline2D(ith) = ir2
  end if
end do
end do

! note: write out the effective input histogram after averaging/alterations.
write(*,*) 'Writing input histogram after averaging/alterations to "input_hist.out"
...
open(91,file='input_hist.out',status='replace')
write(91,*) '# 1. Distance'
write(91,*) '# 2. Cosine Theta'
write(91,*) '# 3. g'
write(91,*) '# 4. f.r'
write(91,*) '# 5. f.s'
write(91,*) '#'
write(91,*) '#'
do ir = 1, histDistBins
  do ith = 1, histCosThBins
    write(91,*) histDist(ir), histCosTh(ith), g2D(ir,ith), fr2D(ir,ith), fs2D(ir,
  ith)
  end do
end do
close(91)

! Spline the log(g) and force arrays using ideal values for the slopes at small r.
This populates the second derivative arrays.
! This requires ideal values that have been averaged over phi.
allocate( idealHist2D(5,histDistBins,histCosThBins) )
call ideal_3D_to_2D(idealHist,histDistBins,histCosThBins,histPhiBins, idealHist2D)

allocate( g2D2(histDistBins,histCosThBins), fr2D2(histDistBins,histCosThBins),
fs2D2(histDistBins,histCosThBins) )
g2D2 = 0_dp; fr2D2 = 0_dp; fs2D2 = 0_dp

do ith = 1, histCosThBins
  call spline(histDist,g2D(:,ith),ispline2D(ith),histDistBins,idealHist2D(2,
ispline2D(ith),ith),real(0,dp), g2D2(:,ith))
  call spline(histDist,fr2D(:,ith),ispline2D(ith),histDistBins,idealHist2D(4,
ispline2D(ith),ith),real(0,dp), fr2D2(:,ith))
  call spline(histDist,fs2D(:,ith),ispline2D(ith),histDistBins,idealHist2D(5,
ispline2D(ith),ith),real(0,dp), fs2D2(:,ith))
end do

!debug
!do ith = histCosThBins, histCosThBins
  !do ir = 1, histDistBins
    !write(45,*) histDist(ir), histCosTh(ith), g2D(ir,ith), g2D2(ir,ith),
idealHist2D(1,ir,ith)
  !end do
!end do
!do ith = histCosThBins, histCosThBins
  !do ir = 1, histDistBins*100
    !xx = ir*(histDistStepSize/100_dp)
    !call splint(histDist,g2D(:,ith),g2D2(:,ith),histDistBins,xx, yy)
    !write(55,*) xx, yy
  !end do
!end do

end subroutine spline_hist_array

```

```

! read LJ--LJ displacements from file
subroutine R_list
  use cfgData
  use ctrlData
  implicit none
  integer          :: ios, i
  character(len=16) :: junk
  character(len=64) :: line

  if (c_explicit_R .eq. 'no') then
    explicit_R = .false.
  else if (c_explicit_R .eq. 'yes') then
    explicit_R = .true.

    ios = 0; crdLines = -1
    open(20,file='crd_list.out',status='old')
    do while(ios>=0)
      read(20,'(a)',IOSTAT=ios) line
      crdLines = crdLines + 1
    end do
    close(20)

    allocate( explicitDist(crdLines) )

    ios = 0
    open(20,file='crd_list.out',status='old')
    do i = 1, crdLines
      read(20,*,iostat=ios) junk, explicitDist(i)
    end do
    close(20)
  end if
end subroutine R_list

! Populate the temporary (cosTh) arrays into a loop.
subroutine set_tmp_arrays
  use histData
  use angleData
  use functions
  implicit none
  integer          :: j
  !real(kind=dp) :: xx,yy !debug

  ! Evaluate the splines for every theta point at these distances and save those
  ! arrays at gTmp1, gTmp2, frTmp1, fsTmp1. The
  ! numbers refer to which solute they correspond to. These arrays are saved and used
  ! until the next distance bin.
  do j = 1, histCosThBins
    if ((rSolv(1).gt.histDist(histDistBins)).and.(rSolv(2).gt.histDist(
histDistBins))) then
      call splint(histDist,g2D(:,j),g2D2(:,j),histDistBins,histDist(histDistBins),
gTmp1(1,j))
      call splint(histDist,g2D(:,j),g2D2(:,j),histDistBins,histDist(histDistBins),
gTmp2(1,j))
      call splint(histDist,fr2D(:,j),fr2D2(:,j),histDistBins,histDist(histDistBins)
, frTmp1(1,j))
      call splint(histDist,fs2D(:,j),fs2D2(:,j),histDistBins,histDist(histDistBins)
, fsTmp1(1,j))
    else if ((rSolv(1).gt.histDist(histDistBins)).and.(rSolv(2).lt.histDist(
histDistBins))) then
      call splint(histDist,g2D(:,j),g2D2(:,j),histDistBins,histDist(histDistBins),
gTmp1(1,j))

```

```

        call splint(histDist,g2D(:,j),g2D2(:,j),histDistBins,rSolvn(2), gTmp2(1,j))
        call splint(histDist,fr2D(:,j),fr2D2(:,j),histDistBins,histDist(histDistBins)
, frTmp1(1,j))
        call splint(histDist,fs2D(:,j),fs2D2(:,j),histDistBins,histDist(histDistBins)
, fsTmp1(1,j))
        else if ((rSolvn(1).lt.histDist(histDistBins)).and.(rSolvn(2).gt.histDist(
histDistBins))) then
            call splint(histDist,g2D(:,j),g2D2(:,j),histDistBins,rSolvn(1), gTmp1(1,j))
            call splint(histDist,g2D(:,j),g2D2(:,j),histDistBins,histDist(histDistBins),
gTmp2(1,j))
            call splint(histDist,fr2D(:,j),fr2D2(:,j),histDistBins,rSolvn(1), frTmp1(1,j)
)
            call splint(histDist,fs2D(:,j),fs2D2(:,j),histDistBins,rSolvn(1), fsTmp1(1,j)
)
        else
            call splint(histDist,g2D(:,j),g2D2(:,j),histDistBins,rSolvn(1), gTmp1(1,j))
            call splint(histDist,g2D(:,j),g2D2(:,j),histDistBins,rSolvn(2), gTmp2(1,j))
            call splint(histDist,fr2D(:,j),fr2D2(:,j),histDistBins,rSolvn(1), frTmp1(1,j)
)
            call splint(histDist,fs2D(:,j),fs2D2(:,j),histDistBins,rSolvn(1), fsTmp1(1,j)
)
        end if
    end do

! Spline the Tmp arrays along theta.
do j = 1, histCosThBins
    call symm_tridag(histCosThStepSize,gTmp1(1,:),histCosThBins, gTmp1(2,:))
    call symm_tridag(histCosThStepSize,gTmp2(1,:),histCosThBins, gTmp2(2,:))
    call symm_tridag(histCosThStepSize,frTmp1(1,:),histCosThBins, frTmp1(2,:))
    call symm_tridag(histCosThStepSize,fsTmp1(1,:),histCosThBins, fsTmp1(2,:))
end do

!debug
!do j = 1, histCosThBins
!write(65,*) histCosTh(j), gTmp1(1,j), gTmp1(2,j)
!end do
!do j = 1, histCosThBins*100
!xx = j*(histCosThStepSize/100_dp)-1_dp
!call symm_splint(histCosTh,histCosThStepSize,gTmp1(1,:),gTmp1(2,:),
histCosThBins,xx, yy)
!write(75,*) xx, yy
!end do
!print*, histCosThStepSize/6_dp

end subroutine set_tmp_arrays

! setup for the average force integral
subroutine setup_compute_avg_force
    use cfgData
    use angleData
    use ctrlData
    implicit none
    integer :: i
    real(kind=dp) :: psiLF

    write(*,*) "Setting up for average force iteration..."

    if (explicit_R .eqv. .true.) then
        cfgRBins = crdLines
        write(*,*) "Number of R Bins: ", cfgRBins
    else if (explicit_R .eqv. .false.) then
        cfgRBins = int( (R_max - R_min)/RStepSize + 1 )
        if (cfgRBins .eq. 0) then
            cfgRBins = 1
        end if
    end if
end subroutine setup_compute_avg_force

```

```

        end if
        write(*,*) "Number of R Bins:      ", cfgRBins
    end if
    xBins = int( (2 * xz_range)/xzStepSize )
    write(*,*) "Number of X Bins:      ", xBins
    zBins = int( (xz_range)/xzStepSize )
    write(*,*) "Number of Z Bins:      ", zBins

    ! allocate array sizes for axes and average force
    allocate( R_axis(cfgRBins), fAvg(cfgRBins), x_axis(xBins), z_axis(zBins) )
    R_axis = 0_dp; fAvg = 0_dp; x_axis = 0_dp; z_axis = 0_dp

    ! allocate arrays for control arrays
    allocate( frcSPA(2, xBins, zBins), grSPA(xBins, zBins) )

    ! Distance Axes
    do i = 1, cfgRBins
        if (explicit_R .eqv. .true.) then
            R_axis(i) = explicitDist(i)
        else if (explicit_r .eqv. .false.) then
            R_axis(i) = (i-1) * RStepSize + R_min
        end if
    end do
    do i = 1, xBins
        x_axis(i) = (i-1) * xzStepSize - xz_range + xzStepSize/2_dp
    end do
    do i = 1, zBins
        z_axis(i) = (i-1) * xzStepSize + xzStepSize/2_dp
    end do

    ! ANGLES
    allocate( cosThetaLF(cfgCosThBins), sinThetaLF(cfgCosThBins), sinPsiLF(cfgPsiBins),
        cosPsiLF(cfgPsiBins) )

    ! Theta
    ! tilt off of z
    cfgCosThStepSize = (cosTh_max - cosTh_min) / real(cfgCosThBins, dp)
    do i = 1, cfgCosThBins
        cosThetaLF(i) = (i-0.5_dp)*cfgCosThStepSize - cosTh_max
        sinThetaLF(i) = sqrt(abs(1_dp-cosThetaLF(i)**2))
    end do
    write(*,*) "Config Cos(Theta) Step Size:      ", cfgCosThStepSize

    ! Psi
    ! processison about z
    cfgPsiStepSize = (psi_max - psi_min) / real(cfgPsiBins, dp)
    do i = 1, cfgPsiBins
        psiLF = (i+0.5_dp)*cfgPsiStepSize
        sinPsiLF(i) = sin(psiLF)
        cosPsiLF(i) = cos(psiLF)
    end do
    write(*,*) "Config Psi Step Size:      ", cfgPsiStepSize
end subroutine setup_compute_avg_force

! do the average force integral
subroutine compute_avg_force
    use cfgData
    use histData
    use angleData
    use ctrlData
    use constants
    use functions
    implicit none

```

```

integer      :: r, i, j, ip, ithLF, ipsiLF, tid, omp_get_thread_num,
             omp_get_num_threads
real(kind=dp)  :: gx, gx2, fx(2)

write(*,*) "Computing average force..."
flush(6)

! Note: until I find a better way to do this. This is how I will allocate the Tmp
arrays.
!$omp PARALLEL DEFAULT( none ) PRIVATE( tid ) SHARED( histCosThBins )
! Allocate temporary wrapped angular arrays for spline interpolation here because
they are THREADPRIVATE and need to be allocated
! for each cpu. The first index determines whether it's the f(x);2nd deriv.
allocate( gTmp1(2,histCosThBins), gTmp2(2,histCosThBins), frTmp1(2,histCosThBins),
fsTmp1(2,histCosThBins) )
!$omp END PARALLEL

! Calculate the average force integral for top half of bisecting plane of cylinder
do r = 1, cfgRBins ! loop lj--lj distances
  frcSPA = 0_dp; grSPA = 0_dp
  !$omp PARALLEL DEFAULT( none ) &
  !$omp PRIVATE( ip, i, j, ithLF, ipsiLF, gx, gx2, fx ) &
  !$omp SHARED( r, xBins, zBins, cut, R_axis, x_axis, z_axis, cfgCosThBins,
cfgPsiBins, histCosTh, histCosThBins, &
  !$omp& histCosThStepSize, cosTh_min, frcSPA, grSPA )
  !!$omp NUM_THREADS( 1 )
  if ((omp_get_thread_num().eq.0).and.(r.eq.1)) then
    write(*,*) 'Parallel CPUs: ', omp_get_num_threads()
    flush(6)
  end if
  !$omp DO SCHEDULE( guided )
  do ip = 1, (xBins*zBins)
    ! Convert single index 'ip' to the x and z indicies 'i' and 'j' respectively.
    i = int((ip-1)/zBins)+1 ! x integer
    j = mod(ip-1,zBins)+1 ! z integer
    !if ((i.eq.19).and.(j.eq.2)) then !debug

    rSolv1(1) = -R_axis(r)/2_dp - x_axis(i)
    rSolv1(2) = 0_dp
    rSolv1(3) = -z_axis(j)
    rSolv1n(1) = euclid_norm(rSolv1)
    rSolv2(1) = R_axis(r)/2_dp - x_axis(i)
    rSolv2(2) = 0_dp
    rSolv2(3) = -z_axis(j)
    rSolv2n(1) = euclid_norm(rSolv2)

    ! Populate and wrap the arrays for taking the derivatives for bicubic
interpolation at this distance.
    call set_tmp_arrays

    ! Loop through orientations of solvent at x(i) and z(j)
    do ithLF = 1, cfgCosThBins
      do ipsiLF = 1, cfgPsiBins
        if ((rSolv1n(1) .lt. 1d-6) .or. (rSolv1n(2) .lt. 1d-6)) then
          gx = 0_dp ! avoid NaNs in calc_angles
        else
          call calc_angles(ipsiLF, ithLF)
          call symm_splint(histCosTh,histCosThStepSize,gTmp1(1,:),gTmp1(2,:),
histCosThBins,cosTh(1), gx) ! solute 1
          call symm_splint(histCosTh,histCosThStepSize,gTmp2(1,:),gTmp2(2,:),
histCosThBins,cosTh(2), gx2) ! solute 2
          gx = exp(gx+gx2)
        end if
      end do
    end do
  end do
end do

```

```

        if (gx .gt. 1d-6) then ! if gx == 0 then don't waste time with the rest
of the calculation
            call symm_splint(histCosTh,histCosThStepSize,frTmp1(1,:),frTmp1(2,:))
, histCosThBins,cosTh(1), fx(1))
            call symm_splint(histCosTh,histCosThStepSize,fsTmp1(1,:),fsTmp1(2,:))
, histCosThBins,cosTh(1), fx(2))

            frcSPA(1,i,j) = frcSPA(1,i,j) + (gx * fx(1) * (-rSolv1(1)/rSolvN(1))
) ! (f.r)*g.R^{hat}
            frcSPA(2,i,j) = frcSPA(2,i,j) + (gx * fx(2) * (-sSolv1(1)/sSolvN(1))
! (f.s)*g.R^{hat}
            grSPA(i,j) = grSPA(i,j) + gx
        end if
    end do !psi
end do !theta
!end if !debug
end do !ip
!$omp END DO
!$omp END PARALLEL

! Add each cell forces to average and normalize
do i = 1, xBins
    do j = 1, zBins
        fAvg(r) = fAvg(r) + ((frcSPA(1,i,j) + frcSPA(2,i,j)) * z_axis(j))
        frcSPA(1,i,j) = frcSPA(1,i,j)/grSPA(i,j)
        frcSPA(2,i,j) = frcSPA(2,i,j)/grSPA(i,j)
        grSPA(i,j) = grSPA(i,j)/cfgCosThBins/cfgPsiBins
    end do !z again
end do !x again
call write_test_out(r) ! write grSPA and frcSPA arrays

! NOTE : After the fact multiply all elements by 2*pi*density/8/pi/pi (2*2pi*pi
/3 (4pi**2)/3 steradians from orientations)
! Number density of chloroform per Angstrom**3 == 0.00750924
fAvg(r) = fAvg(r)/real(2,dp)*density*xzStepSize*xzStepSize*cfgCosThStepSize*
cfgPsiStepSize
end do !r

end subroutine compute_avg_force

! rotate two solvent vectors 'h' for the dipole and 'l' for the Cl1 via a twist 'phi',
tilt 'theta', and procession about z 'psi'
! for lj particles 1 and 2.
subroutine calc_angles(ipsiLF, ithLF)
    use cfgData
    use angleData
    use functions
    use constants
    implicit none
    integer :: ithLF, ipsiLF
    real(kind=dp), dimension(3) :: h

! make rotated solvent dipole vector at origin
h(1) = sinPsiLF(ipsiLF)*sinThetaLF(ithLF)
h(2) = -cosPsiLF(ipsiLF)*sinThetaLF(ithLF)
h(3) = cosThetaLF(ithLF)

! calculate cos(theta1) and cos(theta2) of the solvent to lj-spheres 1 and 2
respectively.
cosTh(1) = dot_product(rSolv1, h) / rSolvN(1)
cosTh(2) = dot_product(rSolv2, h) / rSolvN(2)

tSolv1 = cross_product(rSolv1,h) ! this is (r1 x p) ie. the t vector
sSolv1 = cross_product(tSolv1,rSolv1)

```

```

tSolv1n = euclid_norm(tSolv1)
sSolv1n = euclid_norm(sSolv1)

end subroutine calc_angles

! integrate the average force from 'compute_avg_force' to get the PMF.
subroutine integrate_force
  use cfgData
  use ctrlData
  implicit none
  integer          :: d

  allocate( u_dir(cfgRBins) )
  u_dir = 0_dp

  if (explicit_R .eqv. .false.) then
    do d = 1, cfgRBins
      if (d .eq. 1) then
        u_dir(cfgRBins) = fAvg(cfgRBins) * RStepSize
      else
        u_dir(cfgRBins-(d-1)) = u_dir(cfgRBins-(d-2)) + fAvg(cfgRBins-(d-1)) *
RStepSize
      end if
    end do
  else if (explicit_R .eqv. .true.) then
    do d = 1, cfgRBins
      if (d .eq. 1) then
        u_dir(cfgRBins) = fAvg(cfgRBins) * (R_axis(cfgRBins)-R_axis(cfgRBins-1))
!print*, (R_axis(cfgRBins)-R_axis(cfgRBins-1))
      else
        ! FIXME: is the delta R part of this correct?
        u_dir(cfgRBins-(d-1)) = u_dir(cfgRBins-(d-2)) + fAvg(cfgRBins-(d-1)) * &
(R_axis(cfgRBins-(d-1))-R_axis(cfgRBins-d))
        ! it looks like the first value is getting printed twice. Also, the values
might be wrong. Should it be (d-1) and
! (d-0)? instead of -2 and -1?
!print*, (R_axis(cfgRBins-(d-1))-R_axis(cfgRBins-d))
      end if
    end do
  end if
end subroutine integrate_force

! write force out and g(r) out to compare against explicit
subroutine write_test_out(r)
  use cfgData
  use ctrlData
  implicit none
  integer          :: r, i_f, i, j
  character(len=32) :: temp, filename
  character(len=8)  :: frmt

  i_f = (r-1) * int(RStepSize*10)
  frmt = '(I3.3)' ! an integer of width 3 with zeroes on the left
  write(temp,frmt) i_f ! converting integer to string using 'internal file'
  filename='hist2D_output.'//trim(temp)//'.dat'

  open(35,file=filename,status='replace')
  write(6,*) "Writing test file: ", filename
  write(35,*) "# 1.  X Distance"
  write(35,*) "# 2.  Z Distance"
  write(35,*) "# 3.  g(r)"

```





```

    & fr2(:,:,:), fs2(:,:,:), ft2(:,:,:), gc(:,:,:), gTmp1(:,:,:), gTmp2(:,:,:),
    frTmp1(:,:,:), &
    & fsTmp1(:,:,:), ftTmp1(:,:,:)
real(kind=dp) :: histDistStepSize, histCosThStepSize, histPhiStepSize
integer :: histDistBins, histCosThBins, histPhiBins
integer,allocatable :: ispline(:,:)

!$omp THREADPRIVATE( gTmp1, gTmp2, frTmp1, fsTmp1, ftTmp1 )

end module histData

! data from the config file.
module cfgData
  use prec
  use constants
  real(kind=dp),allocatable :: x_axis(:), z_axis(:), R_axis(:), fAvg(:), u_dir(:)
  real(kind=dp) :: RStepSize, xzStepSize, R_min, R_max, xz_range, cfgCosThStepSize,
  cfgPhiStepSize, cfgPsiStepSize, T, cut, &
  & offset, radius, soluteChg(2)
  character(len=8) :: c_explicit_R
  integer :: cfgRBins, cfgCosThBins, cfgPhiBins, cfgPsiBins
!
  integer :: xBins, zBins
  real(kind=dp) :: density = 0.00750924_dp ! numerical density of chloroforms per
  Angstrom**3
  real(kind=dp) :: cosTh_max = 1_dp
  real(kind=dp) :: cosTh_min = -1_dp
  real(kind=dp) :: phi_max = 2_dp*pi/3_dp ! 2pi/3 is sufficient for a molecule with
  C3 symmetry.
  real(kind=dp) :: phi_hmax = pi/3_dp
  real(kind=dp) :: phi_min = 0_dp
  real(kind=dp) :: psi_max = 2_dp*pi
  real(kind=dp) :: psi_min = 0_dp

end module cfgData

! data for calculating cosTh value.
module angleData
  use prec
  real(kind=dp),allocatable :: sinThetaLF(:), cosThetaLF(:), sinPhiLF(:), cosPhiLF(:)
  , sinPsiLF(:), cosPsiLF(:)
  real(kind=dp) :: rSolv1(3), rSolv2(3), rSolvn(2), cosTh(2), phi(2), sSolv1(3),
  tSolv1(3), sSolvln, tSolvln

  !$omp THREADPRIVATE( rSolv1, rSolv2, rSolvn, sSolv1, sSolvln, tSolv1, tSolvln,
  cosTh, phi )

end module angleData

! testing arrays for force and g(r)
module ctrlData
  use prec
  real(kind=dp),allocatable :: frcSPA(:,:,:), grSPA(:,:), explicitDist(:)
  integer :: crdLines
  logical :: explicit_R

end module ctrlData

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

program compute_avgForce
  use prec

```

```

implicit none
character(len=128) :: histFile, cfgFile, outFile
real(kind=dp) :: omp_get_wtime, ti, tf, seconds
integer :: hours, minutes

ti = omp_get_wtime()

! make list of average direct force from 'collapsed' file.
call parse_command_line(cfgFile) !, outFile)

! read config file
call read_cfg(cfgFile, histFile, outFile)

! make list of average direct force from 'collapsed' file.
call make_hist_table(histFile)

! Now that we have the relevant information spline the g and f arrays along r.
call spline_hist_array

! read in LJ--LJ dist array from file
call R_list

! setup for computing the average force integral.
call setup_compute_avg_force

! compute average force integral.
call compute_avg_force

! integrate average force to get PMF.
call integrate_force

! write PMF output file
call write_output(outFile)

! Write time taken to finish calculation.
tf = omp_get_wtime()

hours = (tf-ti)/3600
minutes = mod((tf-ti),3600d0)/60
seconds = mod(mod((tf-ti),3600d0),60d0)

write(*,*) "~~~~~"
write(*,'(a,i4,a,i2,a,f6.3,a)') "Total time elapsed: ", hours, "h ", minutes, "m "
, seconds, "s"

flush(6)

end program compute_avgForce

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!! Subroutines !!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

! parse commandline for relevant files.
subroutine parse_command_line(cfgFile)
implicit none
character(len=128) :: cfgFile
character(len=16) :: arg
integer :: i
logical :: cfgFileFlag, cfgExist

cfgFileFlag = .false.
cfgExist = .false.
i=1

```

```

do
  call get_command_argument(i,arg)
  select case (arg)

  case ('-cfg')
    i = i+1
    call get_command_argument(i,cfgFile)
    cfgFileFlag=.true.
    INQUIRE(FILE=cfgFile, EXIST=cfgExist)
    write(*,*) 'Config File:          ', cfgFile
    write(*,*) 'Config File Exists:          ', cfgExist
  case default
    write(*,*) 'Unrecognized command-line option: ', arg
    write(*,*) 'Usage: compute_avgForce.x -cfg [cfg file]'
    stop

  end select
  i = i+1
  if (i.ge.command_argument_count()) exit
end do

if (cfgFileFlag.eqv..false.) then
  write(*,*) "Must provide a cfg file using command line argument -cfg [cfg file name]"
  stop
end if

! 'ERROR STOP' if either file doesn't exist
if (cfgExist.eqv..false.) then
  write(*,*) 'cfg file does not exist'
  error stop
end if

flush(6)

end subroutine parse_command_line

! read python cfg file for g(r) parameters
subroutine read_cfg(cfgFile, histFile, outFile)
  use cfgData
  implicit none
  character(len=128) :: cfgFile, histFile, outFile
  character(len=256) :: line
  character(len=128) :: firstWord, sep
  integer :: ios
  logical :: outFileFlag, histFileFlag, histExist, RstepSizeFlag, xzStepSizeFlag,
    RmaxFlag, RminFlag, xzRangeFlag, &
    & thetaBinsFlag, phiBinsFlag, psiBinsFlag, c_explicit_RFlag, TFlag, cutFlag,
    radiusFlag, offsetFlag, soluteChgFlag

  histFileFlag = .false.
  histExist = .false.
  outFileFlag = .false.
  RstepSizeFlag = .false.
  xzStepSizeFlag = .false.
  c_explicit_RFlag = .false.
  RmaxFlag = .false.
  RminFlag = .false.
  xzRangeFlag = .false.
  thetaBinsFlag = .false.
  phiBinsFlag = .false.
  psiBinsFlag = .false.
  TFlag = .false.
  cutFlag = .false.

```

```

radiusFlag = .false.
offsetFlag = .false.
soluteChgFlag = .false.

ios = 0

open(20,file=cfgFile)
do while(ios>=0)
  read(20,'(a)',IOSTAT=ios) line
  call split(line,'=',firstWord, sep)
  if (line .ne. "") then
    if (firstWord .eq. "hist_file") then
      read(line,'(a)') histFile
      write(*,*) "Histogram File:           ", histFile
      histFileFlag = .true.
      INQUIRE(FILE=histFile, EXIST=histExist) ! check if it exists
    else if (firstWord .eq. "out_file") then
      read(line,*) outFile
      write(*,*) "Output File:           ", outFile
      outFileFlag = .true.
    else if (firstWord .eq. "RStepSize") then
      read(line,*) RStepSize
      write(*,*) "PMF Step Size:           ", RStepSize
      RstepSizeFlag = .true.
    else if (firstWord .eq. "xzStepSize") then
      read(line,*) xzStepSize
      write(*,*) "Solvent Grid Step Size:   ", xzStepSize
      xzStepSizeFlag = .true.
    else if (firstWord .eq. "explicit_R") then
      read(line,*) c_explicit_R
      write(*,*) "Use Explicit R Values:    ", c_explicit_R
      c_explicit_RFlag = .true.
    else if (firstWord .eq. "R_max") then
      read(line,*) R_max
      write(*,*) "R Maximum Value:         ", R_max
      RmaxFlag = .true.
    else if (firstWord .eq. "R_min") then
      read(line,*) R_min
      write(*,*) "R Minimum Value:        ", R_min
      RminFlag = .true.
    else if (firstWord .eq. "xz_range") then
      read(line,*) xz_range
      write(*,*) "XZ - Range:             ", xz_range
      xzRangeFlag = .true.
    else if (firstWord .eq. "theta_bins") then
      read(line,*) cfgCosThBins
      write(*,*) "Theta Bins:             ", cfgCosThBins
      thetaBinsFlag= .true.
    else if (firstWord .eq. "phi_bins") then
      read(line,*) cfgPhiBins
      write(*,*) "Phi Bins:               ", cfgPhiBins
      phiBinsFlag= .true.
    else if (firstWord .eq. "psi_bins") then
      read(line,*) cfgPsiBins
      write(*,*) "Psi Bins:               ", cfgPsiBins
      psiBinsFlag= .true.
    else if (firstWord .eq. "temperature") then
      read(line,*) T
      write(*,*) "Temperature (K):        ", T
      TFlag= .true.
    else if (firstWord .eq. "bicubic_cutoff") then
      read(line,*) cut
      write(*,*) "Bicubic/Bilinear Cutoff:  ", cut
      cutFlag= .true.
    else if (firstWord .eq. "solute_radius") then

```

```

        read(line,*) radius
        write(*,*) "Solute radius:      ", radius
        radiusFlag= .true.
    else if (firstWord .eq. "offset") then
        read(line,*) offset
        write(*,*) "Solvent offset distance:      ", offset
        offsetFlag= .true.
    else if (firstWord .eq. "solute_charge") then
        read(line,*) soluteChg(1)
        soluteChg(2) = -soluteChg(1)
        write(*,*) "Solute Charges:      ", soluteChg
        soluteChgFlag= .true.
    end if
end if
end do
close(20)

if (histFileFlag.eqv..false.) then
    write(*,*) "Config file must have a 'hist_file' value"
    stop
end if
if (histExist.eqv..false.) then
    write(*,*) "Config file must point to a 'hist_file' that exists: ", histFile, "
    doesn't exist."
    stop
end if
if (outFileFlag.eqv..false.) then
    write(*,*) "Config file must have a 'out_file' value"
    stop
end if
if (RstepSizeFlag.eqv..false.) then
    write(*,*) "Config file must have a 'RStepSize' value"
    stop
end if
if (xzStepSizeFlag.eqv..false.) then
    write(*,*) "Config file must have a 'xzStepSize' value"
    stop
end if
if (outFileFlag.eqv..false.) then
    write(*,*) "Config file must have a 'out_file' value"
    stop
end if
if (c_explicit_RFlag.eqv..false.) then
    write(*,*) "Config file must have a 'explicit_R' value"
    stop
end if
if (RmaxFlag.eqv..false.) then
    write(*,*) "Config file must have a 'R_max' value"
    stop
end if
if (RminFlag.eqv..false.) then
    write(*,*) "Config file must have a 'R_min' value"
    stop
end if
if (xzRangeFlag.eqv..false.) then
    write(*,*) "Config file must have a 'xz_range' value"
    stop
end if
if (thetaBinsFlag.eqv..false.) then
    write(*,*) "Config file must have a 'theta_bins' value"
    stop
end if
if (phiBinsFlag.eqv..false.) then
    write(*,*) "Config file must have a 'phi_bins' value"
    stop
end if

```

```

end if
if (psiBinsFlag.eqv..false.) then
  write(*,*) "Config file must have a 'psi_bins' value"
  stop
end if
if (TFlag.eqv..false.) then
  write(*,*) "Config file must have a 'temperature' value"
  stop
end if
if (cutFlag.eqv..false.) then
  write(*,*) "Config file must have a 'bicubic_cutoff' value"
  stop
end if
if (radiusFlag.eqv..false.) then
  write(*,*) "Config file must have a 'solute_radius' value"
  stop
end if
if (offsetFlag.eqv..false.) then
  write(*,*) "Config file must have a 'offset' value"
  stop
end if
if (soluteChgFlag.eqv..false.) then
  write(*,*) "Config file must have a 'solute_charge' value"
  stop
end if

flush(6)

end subroutine read_cfg

! read force file and make a lookup table.
subroutine make_hist_table(histFile)
  use histData
  use cfgData
  implicit none
  character(len=128) :: histFile
  character(len=64) :: junk
  character(len=512) :: line
  integer :: ios, ios2, i, j, k, nHistLines
  real(kind=dp),allocatable :: histTmp(:, :)

  ! read number of lines in histFile and allocate that many points in temporary
  histogram list, histTmp.
  ios = 0; nHistLines = -1
  open(20, file=histFile)
  do while(ios>=0)
    read(20, '(a)', IOSTAT=ios) line
    if (line(1:1) .ne. "#") then
      nHistLines = nHistLines + 1
    end if
  end do
  close(20)
  !write(*,*) "nHistLines", nHistLines

  allocate( histTmp(8, nHistLines) )

  ! populate hist arrays
  ios = 0; i = 1
  open(20, file=histFile)
  ! read file ignoring comment lines at the beginning
  do while(ios>=0)
    read(20, '(a)', IOSTAT=ios) line
    if ((line(1:1) .ne. "#") .and. (ios .ge. 0)) then

```

```

!      ! new_format      dist      cos(Th)      phi/3      g(r)+      g(r)-
!      <fLJ.r>+      <fLJ.s>+      <fLJ.t>+
!      read(line,*) histTmp(1,i), histTmp(2,i), histTmp(3,i), histTmp(4,i), junk,
!      histTmp(5,i), histTmp(6,i), histTmp(7,i), &
!      ! fLJ.r- fLJ.s- fLJ.t- fC.r+ fC.s+ fC.t+ fC.r- fC.s- fC.t- gc(r)+
!      gc(r)-
!      & junk, junk, junk, junk, junk, junk, junk, junk, junk, junk, histTmp(8,i
!      ), junk

!      ! old_format      dist      cos(Th)      phi/3      g(r)+      g(r)-
!      <fLJ.r>+      <fLJ.s>+      <fLJ.t>+
!      read(line,*) histTmp(1,i), histTmp(2,i), histTmp(3,i), histTmp(4,i), junk,
!      histTmp(5,i), histTmp(6,i), histTmp(7,i), &
!      ! gc(r)+ gc(r)-
!      & histTmp(8,i), junk
!      i = i + 1
!      end if
!      end do
!      close(20)

! XXX: Unique value determination -- TESTING
do i = 1, nHistLines
  if (i .eq. 1) then
    histDistBins = 1
    ios = 0; ios2 = 0
  else ! i = 2, nHistLines
    if (( histTmp(1,i) .lt. (histTmp(1,i-1)-1d-6) ) .or. ( histTmp(1,i) .gt. (
histTmp(1,i-1)+1d-6) )) then
      ! note: this statement will trigger when a value in the first column (dist
) is different than the value in the row
      ! before it.
      histDistBins = histDistBins + 1
    end if
    if (( histTmp(2,i) .gt. (histTmp(2,1)-1d-6) ) .and. ( histTmp(2,i) .lt. (
histTmp(2,1)+1d-6) ) .and. ( ios .eq. 0 ) &
.and. ( ios2 .eq. 1 )) then
      ! note: this statement will trigger when i = histCosThBins+1 because it
finds the first repeated element
      histCosThBins = (i - 1)/histPhiBins
      ios = 1
    end if
    if (( histTmp(3,i) .gt. (histTmp(3,1)-1d-6) ) .and. ( histTmp(3,i) .lt. (
histTmp(3,1)+1d-6) ) .and. ( ios2 .eq. 0 )) &
then
      ! note: this statement will trigger when i = histPhiBins+1 because it
finds the first repeated element
      histPhiBins = i - 1
      ios2 = 1
    end if
  end if
end do
write(*,*) "Histogram Distance Bins: ", histDistBins
write(*,*) "Histogram Cosine Theta Bins: ", histCosThBins
write(*,*) "Histogram Phi Bins: ", histPhiBins

allocate( histDist(histDistBins), histCosTh(histCosThBins), histPhi(histPhiBins), g
(histDistBins,histCosThBins,histPhiBins), &
& fr(histDistBins,histCosThBins,histPhiBins), fs(histDistBins,histCosThBins,
histPhiBins), &
& ft(histDistBins,histCosThBins,histPhiBins), gc(histDistBins,histCosThBins,
histPhiBins) )

! populate arrays that will be used in the rest of the calculation from temp array
do i = 1, histDistBins ! the values written out from python script are at half
-bin distances

```



```

        histDist(i) = histTmp(1,histCosThBins*histPhiBins*(i-1)+1)
    end do
do i = 1, histCosThBins
    histCosTh(i) = histTmp(2,histPhiBins*(i-1)+1)
end do
do i = 1, histPhiBins
    histPhi(i) = histTmp(3,i)
end do

do i = 1, histDistBins
    do j = 1, histCosThBins
        do k = 1, histPhiBins
            g(i,j,k) = histTmp(4, (i-1)*histCosThBins*histPhiBins + (j-1)*histPhiBins
+ k)
            ! g(r,cos,phi)
            fr(i,j,k) = histTmp(5, (i-1)*histCosThBins*histPhiBins + (j-1)*histPhiBins
+ k)
            ! <f.r>(r,cos,phi)
            fs(i,j,k) = histTmp(6, (i-1)*histCosThBins*histPhiBins + (j-1)*histPhiBins
+ k)
            ! <f.s>(r,cos,phi)
            ft(i,j,k) = histTmp(7, (i-1)*histCosThBins*histPhiBins + (j-1)*histPhiBins
+ k)
            ! <f.t>(r,cos,phi)
            gc(i,j,k) = histTmp(8, (i-1)*histCosThBins*histPhiBins + (j-1)*histPhiBins
+ k)
            ! gc(r,cos,phi)
        end do
    end do
end do

histDistStepSize = histDist(2) - histDist(1)
write(*,*) "Histogram Distance Step Size:      ", histDistStepSize
histCosThStepSize = histCosTh(2) - histCosTh(1)
write(*,*) "Histogram Cosine Theta Step Size:      ", histCosThStepSize
histPhiStepSize = histPhi(2) - histPhi(1)
write(*,*) "Histogram Phi Step Size:                ", histPhiStepSize

flush(6)

end subroutine make_hist_table

! spline the r dimension of each theta phi stack.
subroutine spline_hist_array
    use constants
    use functions
    use histData
    use cfgData
    use idealSolv
    implicit none
    integer                :: i, ir, ith, iphi, imin, igo
    real(kind=dp)          :: x, y, norm_factor
    real(kind=dp),allocatable :: idealHist(:,:,:,:)

    ! Calculate a 4D array idealHist(g/f,r,th,phi)
    allocate( idealHist(2,13,histDistBins,histCosThBins,histPhiBins), ispline(
    histCosThBins,histPhiBins) )
    idealHist = 0_dp; ispline = 0_dp
    call ideal_CL3(histDistBins,histDistStepSize,histCosThBins,cosTh_min,cosTh_max,
    histPhiBins,phi_min,phi_hmax,radius,offset,T, &
    & soluteChg, idealHist)

    ! Edit the input hist arrays to more smoothly transition to -/+ infinity with the
    help of idealHist.
    do ith = 1, histCosThBins
        do iphi = 1, histPhiBins
            imin = 0
            ! Normalization factor or each theta phi array.

```

```

        norm_factor = gc(histDistBins,ith,iphi)/(g(histDistBins,ith,iphi)*4*pi*
histDist(histDistBins)**2)
        ! Find the first non-zero g(r) bin for each theta/phi array and set 'imin' to
that 'ir' index
find:      do ir = 1, histDistBins
            if (g(ir,ith,iphi).gt.1d-6) then
                imin = ir
                exit find
            end if
        end do find

        ! NOTE: Add the ideal values to bins with no sampling. And half counts to
bins that probably should have had sampling.
        igo = 0
        do ir = histDistBins, 1, -1
            if (ir.ge.imin) then
                if (g(ir,ith,iphi).gt.1d-6) then
                    g(ir,ith,iphi) = log(g(ir,ith,iphi))
                else ! note: This is a zero bin where there probably should have been
something. So put a single count in.
                    g(ir,ith,iphi) = log(real(0.5,dp)/(4*pi*histDist(ir)**2)/norm_factor
)

                    fr(ir,ith,iphi) = idealHist(1,2,ir,ith,iphi)
                    fs(ir,ith,iphi) = idealHist(1,3,ir,ith,iphi)
                    ft(ir,ith,iphi) = idealHist(1,4,ir,ith,iphi)
                end if
                else if (ir.lt.imin) then ! .lt.imin ==> in the region of no sampling. Set
the FE (lng) to the direct energy shifted by
                ! a constant energy term , which is the last sampled indirect energy.
                ! ln(g(r<r0)) = -u_dir(r)/T - ( u_pmf(r0)/T - u_dir(r0)/T )
                ! ln(g(r<r0)) = -u_dir(r)/T + ln(g(r0)) - u_dir(r0)/T
                g(ir,ith,iphi) = ( -idealHist(1,1,ir,ith,iphi) + idealHist(1,1,imin,ith
,iphi) ) + g(imin,ith,iphi)
                if ((g(ir,ith,iphi).lt.cut).and.(igo.eq.0)) then ! the largest r to go
past the cutoff
                    ispline(ith,iphi) = ir
                    igo = 1
                end if
                fr(ir,ith,iphi) = idealHist(1,2,ir,ith,iphi)
                if (fr(ir,ith,iphi).gt.-cut) then
                    fr(ir,ith,iphi) = -cut
                end if
                fs(ir,ith,iphi) = idealHist(1,3,ir,ith,iphi)
                if (fs(ir,ith,iphi).gt.-cut) then
                    fs(ir,ith,iphi) = -cut
                end if
                ft(ir,ith,iphi) = idealHist(1,4,ir,ith,iphi)
                if (ft(ir,ith,iphi).gt.-cut) then
                    ft(ir,ith,iphi) = -cut
                end if
            end if
        end do
    end do
end do

! note: write out the effective input histogram after averaging/alterations.
write(*,*) 'Writing input histogram after averaging/alterations to "input_hist.out"
...
open(91,file='input_hist.out',status='replace')
write(91,*) '# 1. Distance'
write(91,*) '# 2. Cosine Theta'
write(91,*) '# 3. Phi [0->pi/3]'
write(91,*) '# 4. g'
write(91,*) '# 5. f.r'
write(91,*) '# 6. f.s'

```

```

write(91,*) '# 7. f.t'
do ir = 1, histDistBins
  do ith = 1, histCosThBins
    do iphi = 1, histPhiBins
      write(91,*) histDist(ir), histCosTh(ith), histPhi(iphi), g(ir,ith,iphi),
        fr(ir,ith,iphi), fs(ir,ith,iphi), &
          & ft(ir,ith,iphi)
    end do
  end do
end do
close(91)

allocate( g2(histDistBins,histCosThBins,histPhiBins), fr2(histDistBins,
  histCosThBins,histPhiBins), &
  & fs2(histDistBins,histCosThBins,histPhiBins), ft2(histDistBins,histCosThBins,
  histPhiBins) )
g2 = 0_dp; fr2 = 0_dp; fs2 = 0_dp; ft2 = 0_dp

! Spline the g and force arrays using ideal values for the slopes at small r. This
! populates the second derivative arrays.
do ith = 1, histCosThBins
  do iphi = 1, histPhiBins
    call spline(histDist,g(:,ith,iphi),ispline(ith,iphi),histDistBins,idealHist
      (1,2,ispline(ith,iphi),ith,iphi),dble(0), &
        & g2(:,ith,iphi))
    call spline(histDist,fr(:,ith,iphi),ispline(ith,iphi),histDistBins,idealHist
      (1,5,ispline(ith,iphi),ith,iphi),dble(0), &
        & fr2(:,ith,iphi))
    call spline(histDist,fs(:,ith,iphi),ispline(ith,iphi),histDistBins,idealHist
      (1,6,ispline(ith,iphi),ith,iphi),dble(0), &
        & fs2(:,ith,iphi))
    call spline(histDist,ft(:,ith,iphi),ispline(ith,iphi),histDistBins,idealHist
      (1,7,ispline(ith,iphi),ith,iphi),dble(0), &
        & ft2(:,ith,iphi))
  end do
end do

end subroutine spline_hist_array

! read LJ--LJ displacements from file
subroutine R_list
  use cfgData
  use ctrlData
  implicit none
  integer          :: ios, i
  character(len=16) :: junk
  character(len=64) :: line

  if (c_explicit_R .eq. 'no') then
    explicit_R = .false.
  else if (c_explicit_R .eq. 'yes') then
    explicit_R = .true.

    ios = 0; crdLines = -1
    open(20,file='crd_list.out',status='old')
    do while(ios>=0)
      read(20,'(a)',IOSTAT=ios) line
      crdLines = crdLines + 1
    end do
    close(20)

    allocate( explicitDist(crdLines) )

    ios = 0

```

```

    open(20, file='crd_list.out', status='old')
    do i = 1, crdLines
        read(20, *, iostat=ios) junk, explicitDist(i)
    end do
    close(20)
end if

end subroutine R_list

! Populate and wrap the temporary (cosTh,phi) arrays into a torus.
subroutine set_tmp_arrays
    use histData
    use angleData
    use functions
    implicit none
    integer                :: j, k

! Evaluate the splines for every theta,phi point at these distances and save those
! arrays at gTmp1, gTmp2, frTmp1, fsTmp1,
! ftTmp1. The numbers refer to which solute they correspond to. These arrays are
! saved and used until the next distance bin.
do j = 1, histCosThBins
    do k = 1, histPhiBins
        if ((rSolv(1).gt.histDist(histDistBins)).and.(rSolv(2).gt.histDist(
histDistBins))) then
            call splint(histDist, g(:, j, k), g2(:, j, k), histDistBins, histDist(histDistBins
-1), gTmp1(1, j, k))
            call splint(histDist, g(:, j, k), g2(:, j, k), histDistBins, histDist(histDistBins
-1), gTmp2(1, j, k))
            call splint(histDist, fr(:, j, k), fr2(:, j, k), histDistBins, histDist(
histDistBins-1), frTmp1(1, j, k))
            call splint(histDist, fs(:, j, k), fs2(:, j, k), histDistBins, histDist(
histDistBins-1), fsTmp1(1, j, k))
            call splint(histDist, ft(:, j, k), ft2(:, j, k), histDistBins, histDist(
histDistBins-1), ftTmp1(1, j, k))
        else if (rSolv(1).gt.histDist(histDistBins)) then
            call splint(histDist, g(:, j, k), g2(:, j, k), histDistBins, histDist(histDistBins
-1), gTmp1(1, j, k))
            call splint(histDist, g(:, j, k), g2(:, j, k), histDistBins, rSolv(2), gTmp2(1, j,
k))
            call splint(histDist, fr(:, j, k), fr2(:, j, k), histDistBins, histDist(
histDistBins-1), frTmp1(1, j, k))
            call splint(histDist, fs(:, j, k), fs2(:, j, k), histDistBins, histDist(
histDistBins-1), fsTmp1(1, j, k))
            call splint(histDist, ft(:, j, k), ft2(:, j, k), histDistBins, histDist(
histDistBins-1), ftTmp1(1, j, k))
        else if (rSolv(2).gt.histDist(histDistBins)) then
            call splint(histDist, g(:, j, k), g2(:, j, k), histDistBins, rSolv(1), gTmp1(1, j,
k))
            call splint(histDist, g(:, j, k), g2(:, j, k), histDistBins, histDist(histDistBins
-1), gTmp2(1, j, k))
            call splint(histDist, fr(:, j, k), fr2(:, j, k), histDistBins, rSolv(1), frTmp1
(1, j, k))
            call splint(histDist, fs(:, j, k), fs2(:, j, k), histDistBins, rSolv(1), fsTmp1
(1, j, k))
            call splint(histDist, ft(:, j, k), ft2(:, j, k), histDistBins, rSolv(1), ftTmp1
(1, j, k))
        else
            call splint(histDist, g(:, j, k), g2(:, j, k), histDistBins, rSolv(1), gTmp1(1, j,
k))
            call splint(histDist, g(:, j, k), g2(:, j, k), histDistBins, rSolv(2), gTmp2(1, j,
k))
            call splint(histDist, fr(:, j, k), fr2(:, j, k), histDistBins, rSolv(1), frTmp1
(1, j, k))

```

```

        call splint(histDist,fs(:,j,k),fs2(:,j,k),histDistBins,rSolvN(1), fsTmp1
(1,j,k))
        call splint(histDist,ft(:,j,k),ft2(:,j,k),histDistBins,rSolvN(1), ftTmp1
(1,j,k))
    end if
    end do
end do
! Note: Wrap g values first before calculating derivatives.
! Wrap all 4 corners
gTmp1(1,0,0) = gTmp1(1,1,1)
gTmp1(1,histCosThBins+1,0) = gTmp1(1,histCosThBins,1)
gTmp1(1,0,histPhiBins+1) = gTmp1(1,1,histPhiBins)
gTmp1(1,histCosThBins+1,histPhiBins+1) = gTmp1(1,histCosThBins,histPhiBins)
gTmp2(1,0,0) = gTmp2(1,1,1)
gTmp2(1,histCosThBins+1,0) = gTmp2(1,histCosThBins,1)
gTmp2(1,0,histPhiBins+1) = gTmp2(1,1,histPhiBins)
gTmp2(1,histCosThBins+1,histPhiBins+1) = gTmp2(1,histCosThBins,histPhiBins)
frTmp1(1,0,0) = frTmp1(1,1,1)
frTmp1(1,histCosThBins+1,0) = frTmp1(1,histCosThBins,1)
frTmp1(1,0,histPhiBins+1) = frTmp1(1,1,histPhiBins)
frTmp1(1,histCosThBins+1,histPhiBins+1) = frTmp1(1,histCosThBins,histPhiBins)
fsTmp1(1,0,0) = fsTmp1(1,1,1)
fsTmp1(1,histCosThBins+1,0) = fsTmp1(1,histCosThBins,1)
fsTmp1(1,0,histPhiBins+1) = fsTmp1(1,1,histPhiBins)
fsTmp1(1,histCosThBins+1,histPhiBins+1) = fsTmp1(1,histCosThBins,histPhiBins)
ftTmp1(1,0,0) = ftTmp1(1,1,1)
ftTmp1(1,histCosThBins+1,0) = ftTmp1(1,histCosThBins,1)
ftTmp1(1,0,histPhiBins+1) = ftTmp1(1,1,histPhiBins)
ftTmp1(1,histCosThBins+1,histPhiBins+1) = ftTmp1(1,histCosThBins,histPhiBins)
! Wrap all 4 edges
gTmp1(1,0, 1:histPhiBins) = gTmp1(1,1, 1:histPhiBins)
gTmp1(1,histCosThBins+1, 1:histPhiBins) = gTmp1(1,histCosThBins, 1:histPhiBins)
gTmp1(1,1:histCosThBins, 0) = gTmp1(1,1:histCosThBins, 1)
gTmp1(1,1:histCosThBins, histPhiBins+1) = gTmp1(1,1:histCosThBins, histPhiBins)
gTmp2(1,0, 1:histPhiBins) = gTmp2(1,1, 1:histPhiBins)
gTmp2(1,histCosThBins+1, 1:histPhiBins) = gTmp2(1,histCosThBins, 1:histPhiBins)
gTmp2(1,1:histCosThBins, 0) = gTmp2(1,1:histCosThBins, 1)
gTmp2(1,1:histCosThBins, histPhiBins+1) = gTmp2(1,1:histCosThBins, histPhiBins)
frTmp1(1,0, 1:histPhiBins) = frTmp1(1,1, 1:histPhiBins)
frTmp1(1,histCosThBins+1, 1:histPhiBins) = frTmp1(1,histCosThBins, 1:histPhiBins)
frTmp1(1,1:histCosThBins, 0) = frTmp1(1,1:histCosThBins, 1)
frTmp1(1,1:histCosThBins, histPhiBins+1) = frTmp1(1,1:histCosThBins, histPhiBins)
fsTmp1(1,0, 1:histPhiBins) = fsTmp1(1,1, 1:histPhiBins)
fsTmp1(1,histCosThBins+1, 1:histPhiBins) = fsTmp1(1,histCosThBins, 1:histPhiBins)
fsTmp1(1,1:histCosThBins, 0) = fsTmp1(1,1:histCosThBins, 1)
fsTmp1(1,1:histCosThBins, histPhiBins+1) = fsTmp1(1,1:histCosThBins, histPhiBins)
ftTmp1(1,0, 1:histPhiBins) = ftTmp1(1,1, 1:histPhiBins)
ftTmp1(1,histCosThBins+1, 1:histPhiBins) = ftTmp1(1,histCosThBins, 1:histPhiBins)
ftTmp1(1,1:histCosThBins, 0) = ftTmp1(1,1:histCosThBins, 1)
ftTmp1(1,1:histCosThBins, histPhiBins+1) = ftTmp1(1,1:histCosThBins, histPhiBins)
! Centered difference method to find derivatives.
! First index: 1 = g(cosTh,phi;R), 2 = dg/dcosTh, 3 = dg/dphi, 4 = d2g/dcosTh/dphi
do j = 1, histCosThBins
    do k = 1, histPhiBins
        gTmp1(2,j,k) = (gTmp1(1,j+1,k)-gTmp1(1,j-1,k))/(2*histCosThStepSize)
        gTmp1(3,j,k) = (gTmp1(1,j,k+1)-gTmp1(1,j,k-1))/(2*histPhiStepSize)
        gTmp1(4,j,k) = (gTmp1(1,j+1,k+1)-gTmp1(1,j+1,k-1)-gTmp1(1,j-1,k+1)+gTmp1(1,j
-1,k-1))/(4*histCosThStepSize*histPhiStepSize)
        gTmp2(2,j,k) = (gTmp2(1,j+1,k)-gTmp2(1,j-1,k))/(2*histCosThStepSize)
        gTmp2(3,j,k) = (gTmp2(1,j,k+1)-gTmp2(1,j,k-1))/(2*histPhiStepSize)
        gTmp2(4,j,k) = (gTmp2(1,j+1,k+1)-gTmp2(1,j+1,k-1)-gTmp2(1,j-1,k+1)+gTmp2(1,j
-1,k-1))/(4*histCosThStepSize*histPhiStepSize)
        frTmp1(2,j,k) = (frTmp1(1,j+1,k)-frTmp1(1,j-1,k))/(2*histCosThStepSize)
        frTmp1(3,j,k) = (frTmp1(1,j,k+1)-frTmp1(1,j,k-1))/(2*histPhiStepSize)

```

```

    frTmp1(4,j,k) = (frTmp1(1,j+1,k+1)-frTmp1(1,j+1,k-1)-frTmp1(1,j-1,k+1)+frTmp1
(1,j-1,k-1)) / &
    & (4*histCosThStepSize*histPhiStepSize)
    fsTmp1(2,j,k) = (fsTmp1(1,j+1,k)-fsTmp1(1,j-1,k))/(2*histCosThStepSize)
    fsTmp1(3,j,k) = (fsTmp1(1,j,k+1)-fsTmp1(1,j,k-1))/(2*histPhiStepSize)
    fsTmp1(4,j,k) = (fsTmp1(1,j+1,k+1)-fsTmp1(1,j+1,k-1)-fsTmp1(1,j-1,k+1)+fsTmp1
(1,j-1,k-1)) / &
    & (4*histCosThStepSize*histPhiStepSize)
    ftTmp1(2,j,k) = (ftTmp1(1,j+1,k)-ftTmp1(1,j-1,k))/(2*histCosThStepSize)
    ftTmp1(3,j,k) = (ftTmp1(1,j,k+1)-ftTmp1(1,j,k-1))/(2*histPhiStepSize)
    ftTmp1(4,j,k) = (ftTmp1(1,j+1,k+1)-ftTmp1(1,j+1,k-1)-ftTmp1(1,j-1,k+1)+ftTmp1
(1,j-1,k-1)) / &
    & (4*histCosThStepSize*histPhiStepSize)
end do
end do

```

```
! xxx: Wrap x1 derivatives
```

```
! Wrap all 4 corners
```

```

gTmp1(2,0,0) = -gTmp1(2,1,1)
gTmp1(2,histCosThBins+1,0) = -gTmp1(2,histCosThBins,1)
gTmp1(2,0,histPhiBins+1) = -gTmp1(2,1,histPhiBins)
gTmp1(2,histCosThBins+1,histPhiBins+1) = -gTmp1(2,histCosThBins,histPhiBins)
gTmp2(2,0,0) = -gTmp2(2,1,1)
gTmp2(2,histCosThBins+1,0) = -gTmp2(2,histCosThBins,1)
gTmp2(2,0,histPhiBins+1) = -gTmp2(2,1,histPhiBins)
gTmp2(2,histCosThBins+1,histPhiBins+1) = -gTmp2(2,histCosThBins,histPhiBins)
frTmp1(2,0,0) = -frTmp1(2,1,1)
frTmp1(2,histCosThBins+1,0) = -frTmp1(2,histCosThBins,1)
frTmp1(2,0,histPhiBins+1) = -frTmp1(2,1,histPhiBins)
frTmp1(2,histCosThBins+1,histPhiBins+1) = -frTmp1(2,histCosThBins,histPhiBins)
fsTmp1(2,0,0) = -fsTmp1(2,1,1)
fsTmp1(2,histCosThBins+1,0) = -fsTmp1(2,histCosThBins,1)
fsTmp1(2,0,histPhiBins+1) = -fsTmp1(2,1,histPhiBins)
fsTmp1(2,histCosThBins+1,histPhiBins+1) = -fsTmp1(2,histCosThBins,histPhiBins)
ftTmp1(2,0,0) = -ftTmp1(2,1,1)
ftTmp1(2,histCosThBins+1,0) = -ftTmp1(2,histCosThBins,1)
ftTmp1(2,0,histPhiBins+1) = -ftTmp1(2,1,histPhiBins)
ftTmp1(2,histCosThBins+1,histPhiBins+1) = -ftTmp1(2,histCosThBins,histPhiBins)
! Wrap all 4 edges
gTmp1(2,0,1:histPhiBins) = -gTmp1(2,1,1:histPhiBins)
gTmp1(2,histCosThBins+1,1:histPhiBins) = -gTmp1(2,histCosThBins,1:histPhiBins)
gTmp1(2,1:histCosThBins,0) = gTmp1(2,1:histCosThBins,1)
gTmp1(2,1:histCosThBins,histPhiBins+1) = gTmp1(2,1:histCosThBins,histPhiBins)
gTmp2(2,0,1:histPhiBins) = -gTmp2(2,1,1:histPhiBins)
gTmp2(2,histCosThBins+1,1:histPhiBins) = -gTmp2(2,histCosThBins,1:histPhiBins)
gTmp2(2,1:histCosThBins,0) = gTmp2(2,1:histCosThBins,1)
gTmp2(2,1:histCosThBins,histPhiBins+1) = gTmp2(2,1:histCosThBins,histPhiBins)
frTmp1(2,0,1:histPhiBins) = -frTmp1(2,1,1:histPhiBins)
frTmp1(2,histCosThBins+1,1:histPhiBins) = -frTmp1(2,histCosThBins,1:histPhiBins)
)
frTmp1(2,1:histCosThBins,0) = frTmp1(2,1:histCosThBins,1)
frTmp1(2,1:histCosThBins,histPhiBins+1) = frTmp1(2,1:histCosThBins,histPhiBins)
fsTmp1(2,0,1:histPhiBins) = -fsTmp1(2,1,1:histPhiBins)
fsTmp1(2,histCosThBins+1,1:histPhiBins) = -fsTmp1(2,histCosThBins,1:histPhiBins)
)
fsTmp1(2,1:histCosThBins,0) = fsTmp1(2,1:histCosThBins,1)
fsTmp1(2,1:histCosThBins,histPhiBins+1) = fsTmp1(2,1:histCosThBins,histPhiBins)
ftTmp1(2,0,1:histPhiBins) = -ftTmp1(2,1,1:histPhiBins)
ftTmp1(2,histCosThBins+1,1:histPhiBins) = -ftTmp1(2,histCosThBins,1:histPhiBins)
)
ftTmp1(2,1:histCosThBins,0) = ftTmp1(2,1:histCosThBins,1)
ftTmp1(2,1:histCosThBins,histPhiBins+1) = ftTmp1(2,1:histCosThBins,histPhiBins)

```

```
! xxx: Wrap x2 derivatives
```

```
! Wrap all 4 corners
```

```

gTmp1(3,0,0) = -gTmp1(3,1,1)
gTmp1(3,histCosThBins+1,0) = -gTmp1(3,histCosThBins,1)
gTmp1(3,0,histPhiBins+1) = -gTmp1(3,1,histPhiBins)
gTmp1(3,histCosThBins+1,histPhiBins+1) = -gTmp1(3,histCosThBins,histPhiBins)
gTmp2(3,0,0) = -gTmp2(3,1,1)
gTmp2(3,histCosThBins+1,0) = -gTmp2(3,histCosThBins,1)
gTmp2(3,0,histPhiBins+1) = -gTmp2(3,1,histPhiBins)
gTmp2(3,histCosThBins+1,histPhiBins+1) = -gTmp2(3,histCosThBins,histPhiBins)
frTmp1(3,0,0) = -frTmp1(3,1,1)
frTmp1(3,histCosThBins+1,0) = -frTmp1(3,histCosThBins,1)
frTmp1(3,0,histPhiBins+1) = -frTmp1(3,1,histPhiBins)
frTmp1(3,histCosThBins+1,histPhiBins+1) = -frTmp1(3,histCosThBins,histPhiBins)
fsTmp1(3,0,0) = -fsTmp1(3,1,1)
fsTmp1(3,histCosThBins+1,0) = -fsTmp1(3,histCosThBins,1)
fsTmp1(3,0,histPhiBins+1) = -fsTmp1(3,1,histPhiBins)
fsTmp1(3,histCosThBins+1,histPhiBins+1) = -fsTmp1(3,histCosThBins,histPhiBins)
ftTmp1(3,0,0) = -ftTmp1(3,1,1)
ftTmp1(3,histCosThBins+1,0) = -ftTmp1(3,histCosThBins,1)
ftTmp1(3,0,histPhiBins+1) = -ftTmp1(3,1,histPhiBins)
ftTmp1(3,histCosThBins+1,histPhiBins+1) = -ftTmp1(3,histCosThBins,histPhiBins)
! Wrap all 4 edges
gTmp1(3,0,1:histPhiBins) = gTmp1(3,1,1:histPhiBins)
gTmp1(3,histCosThBins+1,1:histPhiBins) = gTmp1(3,histCosThBins,1:histPhiBins)
gTmp1(3,1:histCosThBins,0) = -gTmp1(3,1:histCosThBins,1)
gTmp1(3,1:histCosThBins,histPhiBins+1) = -gTmp1(3,1:histCosThBins,histPhiBins)
gTmp2(3,0,1:histPhiBins) = gTmp2(3,1,1:histPhiBins)
gTmp2(3,histCosThBins+1,1:histPhiBins) = gTmp2(3,histCosThBins,1:histPhiBins)
gTmp2(3,1:histCosThBins,0) = -gTmp2(3,1:histCosThBins,1)
gTmp2(3,1:histCosThBins,histPhiBins+1) = -gTmp2(3,1:histCosThBins,histPhiBins)
frTmp1(3,0,1:histPhiBins) = frTmp1(3,1,1:histPhiBins)
frTmp1(3,histCosThBins+1,1:histPhiBins) = frTmp1(3,histCosThBins,1:histPhiBins)
frTmp1(3,1:histCosThBins,0) = -frTmp1(3,1:histCosThBins,1)
frTmp1(3,1:histCosThBins,histPhiBins+1) = -frTmp1(3,1:histCosThBins,histPhiBins)
)
fsTmp1(3,0,1:histPhiBins) = fsTmp1(3,1,1:histPhiBins)
fsTmp1(3,histCosThBins+1,1:histPhiBins) = fsTmp1(3,histCosThBins,1:histPhiBins)
fsTmp1(3,1:histCosThBins,0) = -fsTmp1(3,1:histCosThBins,1)
fsTmp1(3,1:histCosThBins,histPhiBins+1) = -fsTmp1(3,1:histCosThBins,histPhiBins)
)
ftTmp1(3,0,1:histPhiBins) = ftTmp1(3,1,1:histPhiBins)
ftTmp1(3,histCosThBins+1,1:histPhiBins) = ftTmp1(3,histCosThBins,1:histPhiBins)
ftTmp1(3,1:histCosThBins,0) = -ftTmp1(3,1:histCosThBins,1)
ftTmp1(3,1:histCosThBins,histPhiBins+1) = -ftTmp1(3,1:histCosThBins,histPhiBins)
)

! xxx: Wrap cross derivatives
! Wrap all 4 corners
gTmp1(4,0,0) = gTmp1(4,1,1)
gTmp1(4,histCosThBins+1,0) = gTmp1(4,histCosThBins,1)
gTmp1(4,0,histPhiBins+1) = gTmp1(4,1,histPhiBins)
gTmp1(4,histCosThBins+1,histPhiBins+1) = gTmp1(4,histCosThBins,histPhiBins)
gTmp2(4,0,0) = gTmp2(4,1,1)
gTmp2(4,histCosThBins+1,0) = gTmp2(4,histCosThBins,1)
gTmp2(4,0,histPhiBins+1) = gTmp2(4,1,histPhiBins)
gTmp2(4,histCosThBins+1,histPhiBins+1) = gTmp2(4,histCosThBins,histPhiBins)
frTmp1(4,0,0) = frTmp1(4,1,1)
frTmp1(4,histCosThBins+1,0) = frTmp1(4,histCosThBins,1)
frTmp1(4,0,histPhiBins+1) = frTmp1(4,1,histPhiBins)
frTmp1(4,histCosThBins+1,histPhiBins+1) = frTmp1(4,histCosThBins,histPhiBins)
fsTmp1(4,0,0) = fsTmp1(4,1,1)
fsTmp1(4,histCosThBins+1,0) = fsTmp1(4,histCosThBins,1)
fsTmp1(4,0,histPhiBins+1) = fsTmp1(4,1,histPhiBins)
fsTmp1(4,histCosThBins+1,histPhiBins+1) = fsTmp1(4,histCosThBins,histPhiBins)
ftTmp1(4,0,0) = ftTmp1(4,1,1)
ftTmp1(4,histCosThBins+1,0) = ftTmp1(4,histCosThBins,1)

```

```

ftTmp1(4,0,histPhiBins+1) = ftTmp1(4,1,histPhiBins)
ftTmp1(4,histCosThBins+1,histPhiBins+1) = ftTmp1(4,histCosThBins,histPhiBins)
! Wrap all 4 edges
gTmp1(4,0 , 1:histPhiBins) = -gTmp1(4,1 , 1:histPhiBins)
gTmp1(4,histCosThBins+1 , 1:histPhiBins) = -gTmp1(4,histCosThBins , 1:histPhiBins)
gTmp1(4,1:histCosThBins , 0) = -gTmp1(4,1:histCosThBins , 1)
gTmp1(4,1:histCosThBins , histPhiBins+1) = -gTmp1(4,1:histCosThBins , histPhiBins)
gTmp2(4,0 , 1:histPhiBins) = -gTmp2(4,1 , 1:histPhiBins)
gTmp2(4,histCosThBins+1 , 1:histPhiBins) = -gTmp2(4,histCosThBins , 1:histPhiBins)
gTmp2(4,1:histCosThBins , 0) = -gTmp2(4,1:histCosThBins , 1)
gTmp2(4,1:histCosThBins , histPhiBins+1) = -gTmp2(4,1:histCosThBins , histPhiBins)
frTmp1(4,0 , 1:histPhiBins) = -frTmp1(4,1 , 1:histPhiBins)
frTmp1(4,histCosThBins+1 , 1:histPhiBins) = -frTmp1(4,histCosThBins , 1:histPhiBins
)
frTmp1(4,1:histCosThBins , 0) = -frTmp1(4,1:histCosThBins , 1)
frTmp1(4,1:histCosThBins , histPhiBins+1) = -frTmp1(4,1:histCosThBins , histPhiBins
)
fsTmp1(4,0 , 1:histPhiBins) = -fsTmp1(4,1 , 1:histPhiBins)
fsTmp1(4,histCosThBins+1 , 1:histPhiBins) = -fsTmp1(4,histCosThBins , 1:histPhiBins
)
fsTmp1(4,1:histCosThBins , 0) = -fsTmp1(4,1:histCosThBins , 1)
fsTmp1(4,1:histCosThBins , histPhiBins+1) = -fsTmp1(4,1:histCosThBins , histPhiBins
)
ftTmp1(4,0 , 1:histPhiBins) = -ftTmp1(4,1 , 1:histPhiBins)
ftTmp1(4,histCosThBins+1 , 1:histPhiBins) = -ftTmp1(4,histCosThBins , 1:histPhiBins
)
ftTmp1(4,1:histCosThBins , 0) = -ftTmp1(4,1:histCosThBins , 1)
ftTmp1(4,1:histCosThBins , histPhiBins+1) = -ftTmp1(4,1:histCosThBins , histPhiBins
)

```

```
end subroutine set_tmp_arrays
```

```
! setup for the average force integral
```

```
subroutine setup_compute_avg_force
```

```
use cfgData
```

```
use angleData
```

```
use ctrlData
```

```
implicit none
```

```
integer :: i
```

```
real(kind=dp) :: phiLF, psiLF
```

```
write(*,*) "Setting up for average force iteration..."
```

```
if (explicit_R .eqv. .true.) then
```

```
cfgRBins = crdLines
```

```
write(*,*) "Number of R Bins:      ", cfgRBins
```

```
else if (explicit_R .eqv. .false.) then
```

```
cfgRBins = int( (R_max - R_min)/RStepSize + 1 )
```

```
if (cfgRBins .eq. 0) then
```

```
cfgRBins = 1
```

```
end if
```

```
write(*,*) "Number of R Bins:      ", cfgRBins
```

```
end if
```

```
xBins = int( (2 * xz_range)/xzStepSize )
```

```
write(*,*) "Number of X Bins:      ", xBins
```

```
zBins = int( (xz_range)/xzStepSize )
```

```
write(*,*) "Number of Z Bins:      ", zBins
```

```
! allocate array sizes for axes and average force
```

```
allocate( R_axis(cfgRBins), fAvg(cfgRBins), x_axis(xBins), z_axis(zBins) )
```

```
R_axis = 0_dp; fAvg = 0_dp; x_axis = 0_dp; z_axis = 0_dp
```

```
! allocate arrays for control arrays
```

```
allocate( frcSPA(3, xBins, zBins), grSPA(xBins, zBins) )
```



```

! Distance Axes
do i = 1, cfgRBins
  if (explicit_R .eqv. .true.) then
    R_axis(i) = explicitDist(i)
  else if (explicit_r .eqv. .false.) then
    R_axis(i) = (i-1) * RStepSize + R_min
  end if
end do
do i = 1, xBins
  x_axis(i) = (i-1) * xzStepSize - xz_range + xzStepSize/2_dp
end do
do i = 1, zBins
  z_axis(i) = (i-1) * xzStepSize + xzStepSize/2_dp
end do

! ANGLES
allocate( cosThetaLF(cfgCosThBins), sinThetaLF(cfgCosThBins), sinPhiLF(cfgPhiBins),
  cosPhiLF(cfgPhiBins), sinPsiLF(cfgPsiBins), &
  & cosPsiLF(cfgPsiBins) )

! Theta
! tilt off of z
cfgCosThStepSize = (cosTh_max - cosTh_min) / real(cfgCosThBins, dp)
do i = 1, cfgCosThBins
  cosThetaLF(i) = (i-0.5_dp)*cfgCosThStepSize - cosTh_max
  sinThetaLF(i) = sqrt(abs(1_dp-cosThetaLF(i)**2))
end do
write(*,*) "Config Cos(Theta) Step Size:      ", cfgCosThStepSize

! Phi
! twist about z
cfgPhiStepSize = (phi_hmax - phi_min) / real(cfgPhiBins, dp)
do i = 1, cfgPhiBins
  phiLF = (i+0.5_dp)*cfgPhiStepSize
  sinPhiLF(i) = sin(phiLF)
  cosPhiLF(i) = cos(phiLF)
end do
write(*,*) "Config Phi Step Size:          ", cfgPhiStepSize

! Psi
! processison about z
cfgPsiStepSize = (psi_max - psi_min) / real(cfgPsiBins, dp)
do i = 1, cfgPsiBins
  psiLF = (i+0.5_dp)*cfgPsiStepSize
  sinPsiLF(i) = sin(psiLF)
  cosPsiLF(i) = cos(psiLF)
end do
write(*,*) "Config Psi Step Size:          ", cfgPsiStepSize

end subroutine setup_compute_avg_force

! do the average force integral
subroutine compute_avg_force
  use cfgData
  use histData
  use angleData
  use ctrlData
  use constants
  use functions
  implicit none
  integer      :: r, i, j, ip, ithLF, iphiLF, ipsiLF, tid, omp_get_thread_num,
    omp_get_num_threads
  real(kind=dp) :: gx, gx2, fx(3)

```

```

write(*,*) "Computing average force..."
flush(6)

! Note: until I find a better way to do this. This is how I will allocate the Tmp
arrays.
!$omp PARALLEL DEFAULT( none ) PRIVATE( tid ) SHARED( histCosThBins, histPhiBins )
! Allocate temporary wrapped angular arrays for bicubic interpolation here because
they are THREADPRIVATE and need to be
! allocated for each cpu. The first index determines whether it's the f(x);df/dx1;
df/dx2;d2f/dx1dx2.
allocate( gTmp1(4,0:histCosThBins+1,0:histPhiBins+1), gTmp2(4,0:histCosThBins+1,0:
histPhiBins+1), &
& frTmp1(4,0:histCosThBins+1,0:histPhiBins+1), fsTmp1(4,0:histCosThBins+1,0:
histPhiBins+1), &
& ftTmp1(4,0:histCosThBins+1,0:histPhiBins+1) )
!$omp END PARALLEL

! Calculate the average force integral for top half of bisecting plane of cylinder
do r = 1, cfgRBins ! loop lj--lj distances
  frcSPA = 0_dp; grSPA = 0_dp
  !$omp PARALLEL DEFAULT( none ) &
  !$omp PRIVATE( ip, i, j, ithLF, iphiLF, ipsiLF, gx, gx2, fx ) &
  !$omp SHARED( r, xBins, zBins, cut, R_axis, x_axis, z_axis, cfgCosThBins,
cfgPhiBins, cfgPsiBins, histCosTh, histPhi, &
!$omp& histCosThBins, histPhiBins, histCosThStepSize, histPhiStepSize,
cosTh_min, phi_min, frcSPA, grSPA )
  !!$omp NUM_THREADS( 1 )
  if ((omp_get_thread_num().eq.0).and.(r.eq.1)) then
    write(*,*) 'Parallel CPUs: ', omp_get_num_threads()
    flush(6)
  end if
  !$omp DO SCHEDULE( guided )
  do ip = 1, (xBins*zBins)
    ! Convert single index 'ip' to the x and z indicies 'i' and 'j' respectively.
    i = int((ip-1)/zBins)+1
    j = mod(ip-1,zBins)+1

    rSolv1(1) = -R_axis(r)/2_dp - x_axis(i)
    rSolv1(2) = 0_dp
    rSolv1(3) = -z_axis(j)
    rSolv1n(1) = euclid_norm(rSolv1)
    rSolv2(1) = R_axis(r)/2_dp - x_axis(i)
    rSolv2(2) = 0_dp
    rSolv2(3) = -z_axis(j)
    rSolv2n(2) = euclid_norm(rSolv2)

    ! Populate and wrap the arrays for taking the derivatives for bicubic
interpolation at this distance.
    call set_tmp_arrays

    ! Loop through orientations of solvent at x(i) and z(j)
    do ithLF = 1, cfgCosThBins
      do iphiLF = 1, cfgPhiBins
        do ipsiLF = 1, cfgPsiBins
          if ((rSolv1n(1) .lt. 1d-6) .or. (rSolv2n(2) .lt. 1d-6)) then
            gx = 0_dp ! avoid NaNs in calc_angles
          else
            call calc_angles(ipsiLF, ithLF, iphiLF)
            call bicubic_int(cut, histCosTh, histPhi, histCosThBins, histPhiBins,
histCosThStepSize, histPhiStepSize, cosTh_min, &
& phi_min, gTmp1, cosTh(1), phi(1), gx) ! solute 1
            call bicubic_int(cut, histCosTh, histPhi, histCosThBins, histPhiBins,
histCosThStepSize, histPhiStepSize, cosTh_min, &
& phi_min, gTmp2, cosTh(2), phi(2), gx2) ! solute 2
          end if
        end do
      end do
    end do
  end do
end do

```

```

        gx = exp(gx+gx2)
    end if

    if (gx .gt. 1d-6) then ! if gx == 0 then don't waste time with the
rest of the calculation
        call bicubic_int(cut,histCosTh,histPhi,histCosThBins,histPhiBins,
histCosThStepSize,histPhiStepSize,cosTh_min,&
        & phi_min,frTmp1,cosTh(1),phi(1), fx(1))
        call bicubic_int(cut,histCosTh,histPhi,histCosThBins,histPhiBins,
histCosThStepSize,histPhiStepSize,cosTh_min,&
        & phi_min,fsTmp1,cosTh(1),phi(1), fx(2))
        call bicubic_int(cut,histCosTh,histPhi,histCosThBins,histPhiBins,
histCosThStepSize,histPhiStepSize,cosTh_min,&
        & phi_min,ftTmp1,cosTh(1),phi(1), fx(3))

        frcSPA(1,i,j) = frcSPA(1,i,j) + (gx * fx(1) * (-rSolv1(1)/rSolvn
(1))) ! (f.r)*g.R^{hat}
        frcSPA(2,i,j) = frcSPA(2,i,j) + (gx * fx(2) * (-sSolv1(1)/sSolv1n
)) ! (f.s)*g.R^{hat}
        frcSPA(3,i,j) = frcSPA(3,i,j) + (gx * fx(3) * (-tSolv1(1)/tSolv1n
)) ! (f.t)*g.R^{hat}
        grSPA(i,j) = grSPA(i,j) + gx
    end if
end do !psi
end do !phi
end do !theta
end do !ip
!$omp END DO
!$omp END PARALLEL

! Add each cell forces to average and normalize
do i = 1, xBins
do j = 1, zBins
    fAvg(r) = fAvg(r) + ((frcSPA(1,i,j) + frcSPA(2,i,j) + frcSPA(3,i,j)) *
z_axis(j))
    frcSPA(1,i,j) = frcSPA(1,i,j)/grSPA(i,j)
    frcSPA(2,i,j) = frcSPA(2,i,j)/grSPA(i,j)
    frcSPA(3,i,j) = frcSPA(3,i,j)/grSPA(i,j)
    grSPA(i,j) = grSPA(i,j)/cfgCosThBins/cfgPhiBins/cfgPsiBins
end do !z again
end do !x again
call write_test_out(r) ! write grSPA and frcSPA arrays

! NOTE: After the fact multiply all elements by 2*pi*density/8/pi/pi ((2*2pi*pi
/3)=(4pi**2)/3 steradians from orientations)
!     Number density of chloroform per Angstrom**3 == 0.00750924
fAvg(r) = fAvg(r)/real(4,dp)/pi*3_dp*density*xzStepSize*xzStepSize*
cfgCosThStepSize*cfgPhiStepSize*cfgPsiStepSize
end do !r

end subroutine compute_avg_force

! rotate two solvent vectors 'h' for the dipole and 'l' for the Cl1 via a twist 'phi',
tilt 'theta', and procession about z 'psi'
! for lj particles 1 and 2.
subroutine calc_angles(ipsiLF, ithLF, iphiLF)
    use cfgData
    use angleData
    use functions
    use constants
    implicit none
    integer                :: iphiLF, ithLF, ipsiLF
    real(kind=dp),dimension(3)  :: h, x, y

```

```

! make rotated solvent dipole vector at origin
h(1) = sinPsiLF(ipsiLF)*sinThetaLF(ithLF)
h(2) = -cosPsiLF(ipsiLF)*sinThetaLF(ithLF)
h(3) = cosThetaLF(ithLF)

! calculate cos(theta1) and cos(theta2) of the solvent to lj-spheres 1 and 2
  respectively.
cosTh(1) = dot_product(rSolv1, h) / rSolv1n(1)
cosTh(2) = dot_product(rSolv2, h) / rSolv2n(2)

! make rotated vector that represents the x-axis projection of one of the Cl
  vectors.
x(1) = cosPhiLF(iphiLF)*cosPsiLF(ipsiLF) - sinPhiLF(iphiLF)*cosThetaLF(ithLF)*
  sinPsiLF(ipsiLF)
x(2) = cosPhiLF(iphiLF)*sinPsiLF(ipsiLF) + sinPhiLF(iphiLF)*cosThetaLF(ithLF)*
  cosPsiLF(ipsiLF)
x(3) = sinPhiLF(iphiLF)*sinThetaLF(ithLF)

! calculate plane-normal vectors for LJ-C-H and LJ-C-Cl1
! note: use the cross product of the lj-c (rsolv1) and c-h (h) vectors, and the lj-
  c (rsolv1) and c-cl (l) vectors
y = cross_product(h,x)

phi(1) = atan2( dot_product(y,-rSolv1) / (euclid_norm(y)*rSolv1n(1)), dot_product(x
  ,-rSolv1) / (euclid_norm(y)*rSolv1n(1)))
phi(2) = atan2( dot_product(y,-rSolv2) / (euclid_norm(y)*rSolv2n(2)), dot_product(x
  ,-rSolv2) / (euclid_norm(y)*rSolv2n(2)))

! phi [-pi,pi] --> phi'' [0,pi/3]
if ((phi(1) .gt. phi_hmax) .and. (phi(1) .lt. phi_max)) then
  phi(1) = phi_max - phi(1)
else if ((phi(1) .gt. phi_max) .and. (phi(1) .lt. pi)) then
  phi(1) = phi(1) - phi_max
else if ((phi(1) .gt. -pi) .and. (phi(1) .lt. -phi_max)) then
  phi(1) = -(phi(1) + phi_max)
else if ((phi(1) .gt. -phi_max) .and. (phi(1) .lt. -phi_hmax)) then
  phi(1) = phi(1) + phi_max
else if ((phi(1) .gt. -phi_hmax) .and. (phi(1) .lt. phi_min)) then
  phi(1) = -phi(1)
end if
if ((phi(2) .gt. phi_hmax) .and. (phi(2) .lt. phi_max)) then
  phi(2) = phi_max - phi(2)
else if ((phi(2) .gt. phi_max) .and. (phi(2) .lt. pi)) then
  phi(2) = phi(2) - phi_max
else if ((phi(2) .gt. -pi) .and. (phi(2) .lt. -phi_max)) then
  phi(2) = -(phi(2) + phi_max)
else if ((phi(2) .gt. -phi_max) .and. (phi(2) .lt. -phi_hmax)) then
  phi(2) = phi(2) + phi_max
else if ((phi(2) .gt. -phi_hmax) .and. (phi(2) .lt. phi_min)) then
  phi(2) = -phi(2)
end if

tSolv1 = cross_product(rSolv1,h) ! this is (r1 x p) ie. the t vector
sSolv1 = cross_product(tSolv1,rSolv1)
tSolv1n = euclid_norm(tSolv1)
sSolv1n = euclid_norm(sSolv1)

end subroutine calc_angles

! integrate the average force from 'compute_avg_force' to get the PMF.
subroutine integrate_force
  use cfgData
  use ctrlData
  implicit none

```

```

integer      :: d

allocate( u_dir(cfgRBins) )
u_dir = 0_dp

if (explicit_R .eqv. .false.) then
  do d = 1, cfgRBins
    if (d .eq. 1) then
      u_dir(cfgRBins) = fAvg(cfgRBins) * RStepSize
    else
      u_dir(cfgRBins-(d-1)) = u_dir(cfgRBins-(d-2)) + fAvg(cfgRBins-(d-1)) *
RStepSize
    end if
  end do
else if (explicit_R .eqv. .true.) then
  do d = 1, cfgRBins
    if (d .eq. 1) then
      u_dir(cfgRBins) = fAvg(cfgRBins) * (R_axis(cfgRBins)-R_axis(cfgRBins-1))
      !print*, (R_axis(cfgRBins)-R_axis(cfgRBins-1))
    else
      ! FIXME: is the delta R part of this correct?
      u_dir(cfgRBins-(d-1)) = u_dir(cfgRBins-(d-2)) + fAvg(cfgRBins-(d-1)) * &
(R_axis(cfgRBins-(d-1))-R_axis(cfgRBins-d))
      ! it looks like the first value is getting printed twice. Also, the values
might be wrong. Should it be (d-1) and
! (d-0)? instead of -2 and -1?
      !print*, (R_axis(cfgRBins-(d-1))-R_axis(cfgRBins-d))
    end if
  end do
end if

end subroutine integrate_force

! write force out and g(r) out to compare against explicit
subroutine write_test_out(r)
  use cfgData
  use ctrlData
  implicit none
  integer      :: r, i_f, i, j
  character(len=32)  :: temp, filename
  character(len=8)   :: frmt

  i_f = (r-1) * int(RStepSize*10)
  frmt = '(I3.3)' ! an integer of width 3 with zeroes on the left
  write(temp,frmt) i_f ! converting integer to string using 'internal file'
  filename='hist3D_output.'//trim(temp)//'.dat'

  open(35,file=filename,status='replace')
  write(6,*) "Writing test file: ", filename
  write(35,*) "# 1.  X Distance"
  write(35,*) "# 2.  Z Distance"
  write(35,*) "# 3.  g(r)"
  write(35,*) "# 4.  Force.r"
  write(35,*) "# 5.  Force.s"
  write(35,*) "# 6.  Force.t"
  do j = 1, zBins
    do i = 1, xBins
      write(35,898) x_axis(i), z_axis(j), grSPA(i,j), frcSPA(1,i,j), frcSPA(2,i,j),
frcSPA(3,i,j)
    end do
  end do
  close(35)

```



```

end module histData

! data from the config file.
module cfgData
  use prec
  use constants
  real(kind=dp),allocatable :: x_axis(:), z_axis(:), R_axis(:), fAvg(:,,:), u_dir(:,:)
  real(kind=dp) :: RStepSize, xzStepSize, R_min, R_max, xz_range, cfgCosThStepSize,
    cfgPsiStepSize, T, cut, offset, soluteChg(2), radius
  character(len=8) :: c_explicit_R
  integer :: cfgRBins, cfgCosThBins, cfgPhiBins, cfgPsiBins, nThreads
!
  integer :: xBins, zBins
  real(kind=dp) :: density = 0.00750924_dp ! numerical density of chloroforms per
    Angstrom**3
  real(kind=dp) :: dipole_moment = 0.2915_dp ! magnitude of dipole moment chloroform
    in q_e * AA
  real(kind=dp) :: dielectric = 4.39_dp ! reduced dielectric constant of chloroform
  real(kind=dp) :: cosTh_max = 1_dp
  real(kind=dp) :: cosTh_min = -1_dp
  real(kind=dp) :: phi_max = pi/3_dp
  real(kind=dp) :: phi_min = 0_dp
  real(kind=dp) :: psi_max = 2_dp*pi
  real(kind=dp) :: psi_min = 0_dp
end module cfgData

! data for calculating cosTh value.
module angleData
  use prec
  real(kind=dp),allocatable :: sinThetaLF(:), cosThetaLF(:), sinPhiLF(:), cosPhiLF(:)
    , sinPsiLF(:), cosPsiLF(:)
  real(kind=dp) :: rSolv1(3), rSolv2(3), rSolvn(2), cosTh(2), phi(2), sSolv1(3),
    tSolv1(3), sSolv1n, tSolv1n

  !$omp THREADPRIVATE( rSolv1, rSolv2, rSolvn, sSolv1, sSolv1n, tSolv1, tSolv1n,
    cosTh, phi )
end module angleData

! testing arrays for force and g(r)
module ctrlData
  use prec
  real(kind=dp),allocatable :: frcSPA(:, :, :, :), grSPA(:, :, :), explicitDist(:)
  integer :: crdLines
  logical :: explicit_R
end module ctrlData

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!! Main Program !!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

program compute_avgForce
  use prec
  implicit none
  character(len=128) :: histFile, cfgFile, outFile
  real(kind=dp) :: omp_get_wtime, ti, tf, seconds
  integer :: hours, minutes

  ti = omp_get_wtime()

  ! make list of average direct force from 'collapsed' file.
  call parse_command_line(cfgFile)

  ! read config file
  call read_cfg(cfgFile, histFile, outFile)

```

```

! make list of average direct force from 'collapsed' file.
call make_hist_table(histFile)

! Now that we have the relevant information spline the g and f arrays along r.
call spline_hist_array

! read in LJ--LJ dist array from file
call R_list

! setup for computing the average force integral.
call setup_compute_avg_force

! compute average force integral.
call compute_avg_force

! integrate average force to get PMF.
call integrate_force

! write PMF output file
call write_output(outFile)

! Write time taken to finish calculation.
tf = omp_get_wtime()

hours = (tf-ti)/3600
minutes = mod((tf-ti),3600d0)/60
seconds = mod(mod((tf-ti),3600d0),60d0)

write(*,*) "~~~~~"
write(*,'(a,i4,a,i2,a,f6.3,a)') "Total time elapsed:  ", hours, "h ", minutes, "m
", seconds, "s"

flush(6)

end program compute_avgForce

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!! Subroutines !!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

! parse commandline for relevant files.
subroutine parse_command_line(cfgFile)
  implicit none
  character(len=128) :: cfgFile
  character(len=16) :: arg
  integer :: i
  logical :: cfgFileFlag, cfgExist

  cfgFileFlag = .false.
  cfgExist = .false.
  i=1
  do
    call get_command_argument(i,arg)
    select case (arg)

      case ('-cfg')
        i = i+1
        call get_command_argument(i,cfgFile)
        cfgFileFlag=.true.
        INQUIRE(FILE=cfgFile, EXIST=cfgExist)
        write(*,*) 'Config File:          ', cfgFile
        write(*,*) 'Config File Exists:        ', cfgExist
      case default

```



```

        write(*,*) 'Unrecognized command-line option: ', arg
        write(*,*) 'Usage: compute_avgForce.x -cfg [cfg file]'
        stop

    end select
    i = i+1
    if (i.ge.command_argument_count()) exit
end do

if (cfgFileFlag.eqv..false.) then
    write(*,*) "Must provide a cfg file using command line argument -cfg [cfg file name]"
    stop
end if

! 'ERROR STOP' if either file doesn't exist
if (cfgExist.eqv..false.) then
    write(*,*) 'cfg file does not exist'
    error stop
end if

flush(6)

end subroutine parse_command_line

! read python cfg file for g(r) parameters
subroutine read_cfg(cfgFile, histFile, outFile)
    use cfgData
    implicit none
    character(len=128) :: cfgFile, histFile, outFile
    character(len=256) :: line
    character(len=32) :: firstWord, sep
    integer :: ios
    logical :: outFileFlag, histFileFlag, histExist, RstepSizeFlag, xzStepSizeFlag,
        RmaxFlag, RminFlag, xzRangeFlag, &
        & thetaBinsFlag, phiBinsFlag, psiBinsFlag, c_explicit_RFlag, TFlag, cutFlag,
        radiusFlag, offsetFlag, nThreadsFlag, &
        & soluteChgFlag

    histFileFlag = .false.; histExist = .false.
    outFileFlag = .false.
    RstepSizeFlag = .false.
    xzStepSizeFlag = .false.
    c_explicit_RFlag = .false.
    RmaxFlag = .false.
    RminFlag = .false.
    xzRangeFlag = .false.
    thetaBinsFlag = .false.
    phiBinsFlag = .false.
    psiBinsFlag = .false.
    TFlag = .false.
    cutFlag = .false.
    radiusFlag = .false.
    offsetFlag = .false.
    nThreadsFlag = .false.
    soluteChgFlag = .false.

    ios = 0

    open(20, file=cfgFile)
    do while(ios>=0)
        read(20, '(a)', IOSTAT=ios) line
        call split(line, '=', firstWord, sep)
        if (line .ne. "") then

```

```

if (firstWord .eq. "hist_file") then
  read(line,'(a)') histFile
  write(*,*) "Histogram File:          ", histFile
  histFileFlag = .true.
  INQUIRE(FILE=histFile, EXIST=histExist) ! check if it exists
else if (firstWord .eq. "out_file") then
  read(line,*) outFile
  write(*,*) "Output File:                ", outFile
  outFileFlag = .true.
else if (firstWord .eq. "RStepSize") then
  read(line,*) RStepSize
  write(*,*) "PMF Step Size:                ", RStepSize
  RstepSizeFlag = .true.
else if (firstWord .eq. "xzStepSize") then
  read(line,*) xzStepSize
  write(*,*) "Solvent Grid Step Size:      ", xzStepSize
  xzStepSizeFlag = .true.
else if (firstWord .eq. "explicit_R") then
  read(line,*) c_explicit_R
  write(*,*) "Use Explicit R Values:      ", c_explicit_R
  c_explicit_RFlag = .true.
else if (firstWord .eq. "R_max") then
  read(line,*) R_max
  write(*,*) "R Maximum Value:            ", R_max
  RmaxFlag = .true.
else if (firstWord .eq. "R_min") then
  read(line,*) R_min
  write(*,*) "R Minimum Value:           ", R_min
  RminFlag = .true.
else if (firstWord .eq. "xz_range") then
  read(line,*) xz_range
  write(*,*) "XZ - Range:                 ", xz_range
  xzRangeFlag = .true.
else if (firstWord .eq. "theta_bins") then
  read(line,*) cfgCosThBins
  write(*,*) "Theta Bins:                 ", cfgCosThBins
  thetaBinsFlag= .true.
else if (firstWord .eq. "phi_bins") then
  read(line,*) cfgPhiBins
  write(*,*) "Phi Bins:                   ", cfgPhiBins
  phiBinsFlag= .true.
else if (firstWord .eq. "psi_bins") then
  read(line,*) cfgPsiBins
  write(*,*) "Psi Bins:                   ", cfgPsiBins
  psiBinsFlag= .true.
else if (firstWord .eq. "temperature") then
  read(line,*) T
  write(*,*) "Temperature (K):            ", T
  TFlag= .true.
else if (firstWord .eq. "bicubic_cutoff") then
  read(line,*) cut
  write(*,*) "Bicubic/Bilinear Cutoff:    ", cut
  cutFlag= .true.
else if (firstWord .eq. "solute_radius") then
  read(line,*) radius
  write(*,*) "Solute radius:              ", radius
  radiusFlag= .true.
else if (firstWord .eq. "offset") then
  read(line,*) offset
  write(*,*) "Solvent offset distance:    ", offset
  offsetFlag= .true.
else if (firstWord .eq. "num_threads") then
  read(line,*) nThreads
  write(*,*) "Number of Parallel Threads: ", nThreads
  nThreadsFlag= .true.

```

```

        else if (firstWord .eq. "solute_charge") then
            read(line,*) soluteChg(1)
            soluteChg(2) = -soluteChg(1)
            write(*,*) "Solute Charges:      ", soluteChg
            soluteChgFlag= .true.
        end if
    end if
end do
close(20)

if (histFileFlag.eqv..false.) then
    write(*,*) "Config file must have a 'hist_file' value"
    stop
end if
if (histExist.eqv..false.) then
    write(*,*) "Config file must point to a 'hist_file' that exists: ", histFile, "
    doesn't exist."
    stop
end if
if (outFileFlag.eqv..false.) then
    write(*,*) "Config file must have a 'out_file' value"
    stop
end if
if (RstepSizeFlag.eqv..false.) then
    write(*,*) "Config file must have a 'RStepSize' value"
    stop
end if
if (xzStepSizeFlag.eqv..false.) then
    write(*,*) "Config file must have a 'xzStepSize' value"
    stop
end if
if (outFileFlag.eqv..false.) then
    write(*,*) "Config file must have a 'out_file' value"
    stop
end if
if (c_explicit_RFlag.eqv..false.) then
    write(*,*) "Config file must have a 'explicit_R' value"
    stop
end if
if (RmaxFlag.eqv..false.) then
    write(*,*) "Config file must have a 'R_max' value"
    stop
end if
if (RminFlag.eqv..false.) then
    write(*,*) "Config file must have a 'R_min' value"
    stop
end if
if (xzRangeFlag.eqv..false.) then
    write(*,*) "Config file must have a 'xz_range' value"
    stop
end if
if (thetaBinsFlag.eqv..false.) then
    write(*,*) "Config file must have a 'theta_bins' value"
    stop
end if
if (phiBinsFlag.eqv..false.) then
    write(*,*) "Config file must have a 'phi_bins' value"
    stop
end if
if (psiBinsFlag.eqv..false.) then
    write(*,*) "Config file must have a 'psi_bins' value"
    stop
end if
if (TFlag.eqv..false.) then
    write(*,*) "Config file must have a 'temperature' value"

```

```

        stop
    end if
    if (cutFlag.eqv..false.) then
        write(*,*) "Config file must have a 'bicubic_cutoff' value"
        stop
    end if
    if (radiusFlag.eqv..false.) then
        write(*,*) "Config file must have a 'solute_radius' value"
        stop
    end if
    if (offsetFlag.eqv..false.) then
        write(*,*) "Config file must have a 'offset' value"
        stop
    end if
    if (nThreadsFlag.eqv..false.) then
        write(*,*) "Config file must have a 'num_threads' value"
        stop
    end if
    if (soluteChgFlag.eqv..false.) then
        write(*,*) "Config file must have a 'solute_charge' value"
        stop
    end if

    flush(6)

end subroutine read_cfg

! read force file and make a lookup table.
subroutine make_hist_table(histFile)
    use histData; use cfgData
    implicit none
    character(len=128) :: histFile
    character(len=512) :: line
    integer :: ios, ios2, i, j, k, nHistLines
    real(kind=dp),allocatable :: histTmp(:, :)

    ! read number of lines in histFile and allocate that many points in temporary
    histogram list, histTmp.
    ios = 0; nHistLines = -1
    open(20,file=histFile)
    do while(ios>=0)
        read(20,'(a)',IOSTAT=ios) line
        if (line(1:1) .ne. "#") then
            nHistLines = nHistLines + 1
        end if
    end do
    close(20)
    write(*,*) "nHistLines", nHistLines

    allocate( histTmp(19,nHistLines) )

    ! populate hist arrays
    ios = 0; i = 1
    open(20,file=histFile)
    ! read file ignoring comment lines at the beginning
    do while(ios>=0)
        read(20,'(a)',IOSTAT=ios) line
        if ((line(1:1) .ne. "#") .and. (ios .ge. 0)) then
            !
            !           r           cos(Th)           phi/3
            read(line,*) histTmp(1,i), histTmp(2,i), histTmp(3,i), &
            !           g(r)+           g(r)-
            & histTmp(4,i), histTmp(5,i), &
            !           <fLJ.r>+           <fLJ.s>+           <fLJ.t>+
            & histTmp(6,i), histTmp(7,i), histTmp(8,i), &

```

```

! <fLJ.r>- <fLJ.s>- <fLJ.t>-
& histTmp(9,i), histTmp(10,i), histTmp(11,i), &
! <fC.r>+ <fC.s>+ <fC.t>+
& histTmp(12,i), histTmp(13,i), histTmp(14,i), &
! <fC.r>- <fC.s>- <fC.t>-
& histTmp(15,i), histTmp(16,i), histTmp(17,i), &
! gc(r)+ gc(r)-
& histTmp(18,i), histTmp(19,i)

i = i + 1
end if
end do
close(20)

! Unique value determination
do i = 1, nHistLines
if (i .eq. 1) then
histDistBins = 1
ios = 0; ios2 = 0
else ! i = 2, nHistLines
if (( histTmp(1,i) .lt. (histTmp(1,i-1)-1d-6) ) .or. ( histTmp(1,i) .gt. (
histTmp(1,i-1)+1d-6) )) then
! note: this statement will trigger when a value in the first column (dist
) is different than the value in the row
! before it.
histDistBins = histDistBins + 1
end if
if (( histTmp(2,i) .gt. (histTmp(2,1)-1d-6) ) .and. ( histTmp(2,i) .lt. (
histTmp(2,1)+1d-6) ) .and. ( ios .eq. 0 ) .and. &
& ( ios2 .eq. 1 ) ) then
! note: this statement will trigger when i = histCosThBins+1 because it
finds the first repeated element
histCosThBins = (i - 1)/histPhiBins
ios = 1
end if
if (( histTmp(3,i) .gt. (histTmp(3,1)-1d-6) ) .and. ( histTmp(3,i) .lt. (
histTmp(3,1)+1d-6) ) .and. ( ios2 .eq. 0 )) then
! note: this statement will trigger when i = histPhiBins+1 because it
finds the first repeated element
histPhiBins = i - 1
ios2 = 1
end if
end if
end do
write(*,*) "Histogram Distance Bins: ", histDistBins
write(*,*) "Histogram Cosine Theta Bins: ", histCosThBins
write(*,*) "Histogram Phi Bins: ", histPhiBins

allocate( histDist(histDistBins), histCosTh(histCosThBins), histPhi(histPhiBins), g
(2,histDistBins,histCosThBins,histPhiBins), &
& fLJr(2,histDistBins,histCosThBins,histPhiBins), fLJs(2,histDistBins,
histCosThBins,histPhiBins), &
& fLJt(2,histDistBins,histCosThBins,histPhiBins), fCr(2,histDistBins,
histCosThBins,histPhiBins), &
& fCs(2,histDistBins,histCosThBins,histPhiBins), fCt(2,histDistBins,
histCosThBins,histPhiBins), &
& gc(2,histDistBins,histCosThBins,histPhiBins) )

! populate arrays that will be used in the rest of the calculation from temp array
do i = 1, histDistBins ! the values written out from python script are at half
-bin distances
histDist(i) = histTmp(1,histCosThBins*histPhiBins*(i-1)+1)
end do
do i = 1, histCosThBins
histCosTh(i) = histTmp(2,histPhiBins*(i-1)+1)
end do

```

```

do i = 1, histPhiBins
    histPhi(i) = histTmp(3,i)
end do

do i = 1, histDistBins
    do j = 1, histCosThBins
        do k = 1, histPhiBins
            g(1,i,j,k) = histTmp( 4, (i-1)*histCosThBins*histPhiBins + (j-1)*
histPhiBins + k) ! g(r,cos,phi)+ currently g
            g(2,i,j,k) = histTmp( 5, (i-1)*histCosThBins*histPhiBins + (j-1)*
histPhiBins + k) ! g(r,cos,phi)- currently g
            fLJr(1,i,j,k) = histTmp( 6, (i-1)*histCosThBins*histPhiBins + (j-1)*
histPhiBins + k) ! <fLJ.r>(r,cos,phi) +
            fLJs(1,i,j,k) = histTmp( 7, (i-1)*histCosThBins*histPhiBins + (j-1)*
histPhiBins + k) ! <fLJ.s>(r,cos,phi) +
            fLJt(1,i,j,k) = histTmp( 8, (i-1)*histCosThBins*histPhiBins + (j-1)*
histPhiBins + k) ! <fLJ.t>(r,cos,phi) +
            fLJr(2,i,j,k) = histTmp( 9, (i-1)*histCosThBins*histPhiBins + (j-1)*
histPhiBins + k) ! <fLJ.r>(r,cos,phi) -
            fLJs(2,i,j,k) = histTmp(10, (i-1)*histCosThBins*histPhiBins + (j-1)*
histPhiBins + k) ! <fLJ.s>(r,cos,phi) -
            fLJt(2,i,j,k) = histTmp(11, (i-1)*histCosThBins*histPhiBins + (j-1)*
histPhiBins + k) ! <fLJ.t>(r,cos,phi) -
            fCr(1,i,j,k) = histTmp(12, (i-1)*histCosThBins*histPhiBins + (j-1)*
histPhiBins + k) ! <fC.r>(r,cos,phi) +
            fCs(1,i,j,k) = histTmp(13, (i-1)*histCosThBins*histPhiBins + (j-1)*
histPhiBins + k) ! <fC.s>(r,cos,phi) +
            fCt(1,i,j,k) = histTmp(14, (i-1)*histCosThBins*histPhiBins + (j-1)*
histPhiBins + k) ! <fC.t>(r,cos,phi) +
            fCr(2,i,j,k) = histTmp(15, (i-1)*histCosThBins*histPhiBins + (j-1)*
histPhiBins + k) ! <fC.r>(r,cos,phi) -
            fCs(2,i,j,k) = histTmp(16, (i-1)*histCosThBins*histPhiBins + (j-1)*
histPhiBins + k) ! <fC.s>(r,cos,phi) -
            fCt(2,i,j,k) = histTmp(17, (i-1)*histCosThBins*histPhiBins + (j-1)*
histPhiBins + k) ! <fC.t>(r,cos,phi) -
            gc(1,i,j,k) = histTmp(18, (i-1)*histCosThBins*histPhiBins + (j-1)*
histPhiBins + k) ! gc(r,cos,phi) +
            gc(2,i,j,k) = histTmp(19, (i-1)*histCosThBins*histPhiBins + (j-1)*
histPhiBins + k) ! gc(r,cos,phi) +
        end do
    end do
end do

histDistStepSize = histDist(2) - histDist(1)
write(*,*) "Histogram Distance Step Size:      ", histDistStepSize
histCosThStepSize = histCosTh(2) - histCosTh(1)
write(*,*) "Histogram Cosine Theta Step Size:      ", histCosThStepSize
histPhiStepSize = histPhi(2) - histPhi(1)
write(*,*) "Histogram Phi Step Size:          ", histPhiStepSize

flush(6)
!print*, 'look4', g(1,30,1,1), g(2,30,1,1)

end subroutine make_hist_table

! calculate reduced mean field emf1D(r) from g(r,cosTh,phi)
subroutine reduced_mean_field(ia)
    use histData; use functions
    implicit none
    integer :: ia, ir, ith, iphi
    real(kind=dp) :: boltz, boltz_sum
    real(kind=dp), allocatable :: u02D(:, :)
    ! real(kind=dp) :: x, y1, y2 !debug

```

```

write(*,*) 'Calculating reduced mean field...'; flush(6)

allocate( u02D(histDistBins,histCosThBins) )
u02D = 0_dp
u02D = minval(-g(ia,::,::),dim=3)

avgCosTh(ia,:) = 0_dp
do ir = 1, histDistBins
  boltz_sum = 0_dp
  do ith = 1, histCosThBins
    do iphi = 1, histPhiBins
      boltz = exp(g(ia,ir,ith,iphi) !- u02D(ir,ith)) ! i think this should be a
+ sign but the data looks wrong when it is.
      avgCosTh(ia,ir) = avgCosTh(ia,ir) + (boltz * histCosTh(ith))
      boltz_sum = boltz_sum + boltz
    end do
  end do
  if (abs(boltz_sum).lt.1.0d-6) then
    avgCosTh(ia,ir) = 0_dp
  else
    avgCosTh(ia,ir) = avgCosTh(ia,ir) / boltz_sum
  end if
end do

do ir = 1, histDistBins
  emf1D(ia,ir) = (3*avgCosTh(ia,ir) - avgCosTh(ia,ir)*(6*avgCosTh(ia,ir)**2 +
  avgCosTh(ia,ir)**4 - 2*avgCosTh(ia,ir)**6)/5.) &
  & / (1-avgCosTh(ia,ir)**2) ! inverse Langevin function
end do

if (ia.eq.2) then !debug
  open(99,file='emf.out',status='replace')
  write(99,*) '# 1. Distance'
  write(99,*) '# 2. emf +'
  write(99,*) '# 3. <p> +'
  write(99,*) '# 4. emf -'
  write(99,*) '# 5. <p> -'
  !do ir = 1, histDistBins*100
  !x = ir/100.0/10.0
  !call splint(histDist,emf1D(1,:),emf1D2(1,:),histDistBins,x, y1)
  !call splint(histDist,emf1D(2,:),emf1D2(2,:),histDistBins,x, y2)
  !write(99,*) x, y1, y2
  !end do
  do ir = 1, histDistBins
    write(99,*) histDist(ir), emf1D(1,ir), avgCosTh(1,ir), emf1D(2,ir), avgCosTh
(2,ir)
  end do
  close(99)
end if
end subroutine reduced_mean_field

! spline the r dimension of each theta phi stack and then average over phi for 2D
subroutine spline_hist_array
  use constants; use functions; use histData; use cfgData; use idealSolv
  implicit none
  integer :: ia, ir, ith, iphi, imin, igo, igo1, igo2
  real(kind=dp) :: norm_factor, boltz, boltz_sum
  real(kind=dp),allocatable :: idealHist(:,::,::,::), idealHist2D(:,::,::,::), u02D(:,::),
  idealHist1D(:,::,::), u01D(:)
  integer,allocatable :: ispline(:,::), ispline2D(:)
  integer :: ispline1D
  !real(kind=dp) :: xx, yy !debug
  !integer :: rTmp !debug

```

```

write(*,*) 'Editing input histogram arrays with ideal arrays in 3D...'

! Calculate a 4D array idealHist(lj+/lj-,g/f,r,th,phi)
allocate( idealHist(2,13,histDistBins,histCosThBins,histPhiBins), ispline(
histCosThBins,histPhiBins), ispline2D(histCosThBins) )
idealHist = 0_dp; ispline = 0_dp
call ideal_CL3(histDistBins,histDistStepSize,histCosThBins,cosTh_min,cosTh_max,
histPhiBins,phi_min,phi_max,radius,offset,T, &
& soluteChg, idealHist)
!print*, 'look1', idealHist(1,1,30,1,1), idealHist(2,1,30,1,1) !debug

! Spline the log(g) and force arrays using ideal values for the slopes at small r.
This populates the second derivative arrays.
! This requires ideal values that have been averaged over phi.
allocate( idealHist2D(2,9,histDistBins,histCosThBins) )
idealHist2D = 0_dp
call ideal_3D_to_2D(idealHist,histDistBins,histCosThBins,histPhiBins, idealHist2D)

allocate( idealHist1D(2,5,histDistBins) )
idealHist1D = 0_dp
call ideal_2D_to_1D(idealHist2D,histDistBins,histCosThBins, idealHist1D)

! Allocate explicit data arrays for 3D -> 2D transformation
allocate( g2D(2,histDistBins,histCosThBins), fLJr2D(2,histDistBins,histCosThBins),
fLJs2D(2,histDistBins,histCosThBins), &
& fCr2D(2,histDistBins,histCosThBins), fCs2D(2,histDistBins,histCosThBins), u02D
(histDistBins,histCosThBins) )

g2D = 0_dp; fLJr2D = 0_dp; fLJs2D = 0_dp; fCr2D = 0_dp; fCs2D = 0_dp; u02D = 0_dp

! Allocate explicit data arrays for 2D -> 1D transformation
allocate( g1D(2,histDistBins), fLJr1D(2,histDistBins), fCr1D(2,histDistBins), u01D(
histDistBins) )
g1D = 0_dp; fLJr1D = 0_dp; fCr1D = 0_dp; u01D = 0_dp

allocate( g1D2(2,histDistBins), fLJr1D2(2,histDistBins), fCr1D2(2,histDistBins),
emf1D(2,histDistBins), &
& emf1D2(2,histDistBins), avgCosTh(2,histDistBins) )
g1D2 = 0_dp; fLJr1D2 = 0_dp; fCr1D2 = 0_dp; emf1D = 0_dp; emf1D2 = 0_dp

! Edit the input hist arrays to more smoothly transition to -/+ infinity with the
help of idealHist.
do ia = 1, 2 ! which solute
do ith = 1, histCosThBins
do iphi = 1, histPhiBins
imin = 0
! Normalization factor for each theta phi array.
norm_factor = gc(ia,histDistBins,ith,iphi)/(g(ia,histDistBins,ith,iphi)*4*
pi*histDist(histDistBins)**2)
! Find the first non-zero g(r) bin for each theta/phi array and set 'imin'
to that 'ir' index
find: do ir = 1, histDistBins
if (g(ia,ir,ith,iphi).gt.1d-6) then
imin = ir
exit find
end if
end do find

! Note: Add the ideal values to bins with no sampling. And half counts to
bins that probably should have had sampling.
igo = 0
do ir = histDistBins, 1, -1
if (ir.ge.imin) then

```



```

        if (g(ia,ir,ith,iphi).gt.1d-6) then
            g(ia,ir,ith,iphi) = log(g(ia,ir,ith,iphi)) ! g is ln(g) now
        else ! note: This is a zero bin where there probably should have
            been something. So put a half count in.
            g(ia,ir,ith,iphi) = log(real(0.5,dp)/(4*pi*histDist(ir)**2)/
norm_factor)
            fLJr(ia,ir,ith,iphi) = idealHist(ia, 2,ir,ith,iphi)
            fLJs(ia,ir,ith,iphi) = idealHist(ia, 3,ir,ith,iphi)
            fLJt(ia,ir,ith,iphi) = idealHist(ia, 4,ir,ith,iphi)
            fCr(ia,ir,ith,iphi) = idealHist(ia, 8,ir,ith,iphi)
            fCs(ia,ir,ith,iphi) = idealHist(ia, 9,ir,ith,iphi)
            fCt(ia,ir,ith,iphi) = idealHist(ia,10,ir,ith,iphi)
        end if
        else if (ir.lt.imin) then ! Analytic Continuation: .lt.imin ==> in the
            region of no sampling. Set the FE (log(g)) to
            ! the direct energy shifted by a constant energy term , the
            difference between the last sampled indirect and
            ! direct energies.
            ! ln(g(r<r0)) = -u_dir(r)/T - ( u_pmf(r0)/T - u_dir(r0)/T )
            ! ln(g(r<r0)) = -u_dir(r)/T + ln(g(r0)) - u_dir(r0)/T
            g(ia,ir,ith,iphi) = ( -idealHist(ia,1,ir,ith,iphi) + idealHist(ia,1,
imin,ith,iphi) ) + g(ia,imin,ith,iphi) ! ln(g)
            !print*, 'look2', idealHist(ia,1,30,1,1) !debug
            if ((g(ia,ir,ith,iphi).lt.cut).and.(igo.eq.0)) then ! the largest r
to go past the cutoff
                ispline(ith,iphi) = ir
                igo = 1
            end if
            fLJr(ia,ir,ith,iphi) = idealHist(ia, 2,ir,ith,iphi)
            fLJs(ia,ir,ith,iphi) = idealHist(ia, 3,ir,ith,iphi)
            fLJt(ia,ir,ith,iphi) = idealHist(ia, 4,ir,ith,iphi)
            fCr(ia,ir,ith,iphi) = idealHist(ia, 8,ir,ith,iphi)
            fCs(ia,ir,ith,iphi) = idealHist(ia, 9,ir,ith,iphi)
            fCt(ia,ir,ith,iphi) = idealHist(ia,10,ir,ith,iphi)
        end if
    end do ! ir
end do ! iphi
end do ! ith

! Set all ln(g) values past the largest r bin to reach the cutoff to the
cutoff value.
igo = 0
do iphi = 1, histPhiBins
    do ith = 1, histCosThBins
        do ir = histDistBins, 1, -1
            if ((g(ia,ir,ith,iphi).lt.-abs(cut)).and.(igo.eq.0)) then
                g(ia,1:ir,ith,iphi) = -abs(cut)
                igo = 1
            end if
        end do
    end do
    igo = 0
end do
end do

call reduced_mean_field(ia) ! this way the emf1D is made with the already
analytically continued g(r,th,phi) which is ln(g)

! Set all LJ forces past (to the left of) the first (largest r) bin to reach the
cutoff to the cutoff value.
igo = 0; igo1 = 0; igo2 = 0
do iphi = 1, histPhiBins
    do ith = 1, histCosThBins
        do ir = histDistBins, 1, -1
            if ((fLJr(ia,ir,ith,iphi).gt.abs(cut)).and.(igo.eq.0)) then
                fLJr(ia,1:ir,ith,iphi) = abs(cut)
            end if
        end do
    end do
end do

```

```

        igo = 1
    end if
    if ((fLJs(ia,ir,ith,iphi).gt.abs(cut)).and.(igo1.eq.0)) then
        fLJs(ia,1:ir,ith,iphi) = abs(cut)
        igo1 = 1
    end if
    if ((fLJt(ia,ir,ith,iphi).gt.abs(cut)).and.(igo2.eq.0)) then
        fLJt(ia,1:ir,ith,iphi) = abs(cut)
        igo2 = 1
    end if
end do
igo = 0; igo1 = 0; igo2 = 0
end do
end do

! Set all C forces past (to the left of) the first (largest r) bin to reach the
cutoff to the cutoff value.
igo = 0; igo1 = 0; igo2 = 0
do iphi = 1, histPhiBins
    do ith = 1, histCosThBins
        do ir = histDistBins, 1, -1
            if ((abs(fCr(ia,ir,ith,iphi)).gt.abs(cut)).and.(igo.eq.0)) then
                fCr(ia,1:ir,ith,iphi) = sign(real(1,dp),idealHist(ia,8,ir,ith,iphi))
* abs(cut)
                igo = 1
            end if
            if ((abs(fCs(ia,ir,ith,iphi)).gt.abs(cut)).and.(igo1.eq.0)) then
                fCs(ia,1:ir,ith,iphi) = sign(real(1,dp),idealHist(ia,9,ir,ith,iphi))
* abs(cut)
                igo1 = 1
            end if
            if ((abs(fCt(ia,ir,ith,iphi)).gt.abs(cut)).and.(igo2.eq.0)) then
                fCt(ia,1:ir,ith,iphi) = sign(real(1,dp),idealHist(ia,10,ir,ith,iphi))
) * abs(cut)
                igo2 = 1
            end if
        end do
        igo = 0; igo1 = 0; igo2 = 0
    end do
end do
! do ir = 1, histDistBins !debug
! print*, 'g3D: ', ir, g(ia,ir,1,1) !debug
! end do !debug

! Average the 3D input histograms into 2D
write(*,*) 'Averaging input histograms from 3D into 2D for solute: ',ia

! Find the minimum value of u(phi; r,th) ==> u02D(r,th)
! dim=3 in this case means the phi dimension. Replace the array in phi at each r
,th with the minimum value of the array,
! making an array u02D(r,th).
u02D = minval(-g(ia,:::,:::),dim=3)
! do ir = 1, histDistBins !debug
! print*, 'u0_2D: ',ir, u02D(ir,1) !debug
! end do !debug

do ith = 1, histCosThBins
    do ir = 1, histDistBins
        boltz_sum = 0_dp
        do iphi = 1, histPhiBins
            boltz = exp(g(ia,ir,ith,iphi) + u02D(ir,ith))
            !print*, 'boltz: ', g(ia,ir,ith,iphi), u02D(ir,ith), boltz !debug
            g2D(ia,ir,ith) = g2D(ia,ir,ith) + exp(g(ia,ir,ith,iphi)) ! g is g
            !print*, 'g2Dloop: ', ith,ir,iphi, g2D(ia,ir,ith) !debug
        end do
    end do
end do

```

```

! fLJ.r      fLJr2D(ia,ir,ith) = fLJr2D(ia,ir,ith) + (boltz * fLJr(ia,ir,ith,iphi))
! fLJ.s      fLJs2D(ia,ir,ith) = fLJs2D(ia,ir,ith) + (boltz * fLJs(ia,ir,ith,iphi))
! fLJ.s      fCr2D(ia,ir,ith) = fCr2D(ia,ir,ith) + (boltz * fCr(ia,ir,ith,iphi)) !
fC.r        fCs2D(ia,ir,ith) = fCs2D(ia,ir,ith) + (boltz * fCs(ia,ir,ith,iphi)) !
fC.s        boltz_sum = boltz_sum + boltz ! denominator for averaging over phi
            end do
            !print*, 'boltz_sum: ', boltz_sum !debug
            g2D(ia,ir,ith) = log(g2D(ia,ir,ith) / real(histPhiBins,dp)) ! finish
average over phi by dividing and converting to log(g)
            if (g2D(ia,ir,ith).lt.-abs(cut)) g2D(ia,ir,ith) = -abs(cut)
            fLJr2D(ia,ir,ith) = fLJr2D(ia,ir,ith) / boltz_sum
            fLJs2D(ia,ir,ith) = fLJs2D(ia,ir,ith) / boltz_sum
            fCr2D(ia,ir,ith) = fCr2D(ia,ir,ith) / boltz_sum
            fCs2D(ia,ir,ith) = fCs2D(ia,ir,ith) / boltz_sum
            end do
            !print*, ith, fLJr2D(ia,1:40,ith) !debug
        end do
!         do ir = 1, histDistBins !debug
!             print*, 'g2D: ', ir, g2D(ia,ir,1) !debug
!         end do !debug

! Average the 2D input histograms into 1D
write(*,*) 'Averaging input histograms from 2D into 1D for solute: ',ia

u01D(:) = minval(-g2D(ia,.,:),dim=2)
!         do ir = 1, histDistBins !debug
!             print*, 'u0_1D: ',ir, u01D(ir) !debug
!         end do !debug

do ir = 1, histDistBins
    boltz_sum = 0_dp
    do ith = 1, histCosThBins
        boltz = exp(g2D(ia,ir,ith) + u01D(ir))
        !print*, 'boltz: ', g2D(ia,ir,ith), u01D(ir), boltz !debug
        g1D(ia,ir) = g1D(ia,ir) + exp(g2D(ia,ir,ith)) ! g1D is g
        fLJr1D(ia,ir) = fLJr1D(ia,ir) + (boltz * fLJr2D(ia,ir,ith)) ! fLJ.r
        fCr1D(ia,ir) = fCr1D(ia,ir) + (boltz * fCr2D(ia,ir,ith)) ! fC.r
        boltz_sum = boltz_sum + boltz ! denominator for averaging over theta
    end do
    !print*, 'boltz_sum: ', boltz_sum !debug
    g1D(ia,ir) = log(g1D(ia,ir) / real(histCosThBins,dp)) ! finish average over
cosTh by dividing and converting to log(g)
    fLJr1D(ia,ir) = fLJr1D(ia,ir) / boltz_sum
    fCr1D(ia,ir) = fCr1D(ia,ir) / boltz_sum
    end do
    !print*, 'fLJ1D: ', fLJr1D(ia,1:40) !debug
! After the averaging is done enforce the cutoff
do ir = 1, histDistBins
    if (g1D(ia,ir).lt.cut) then
        g1D(ia,ir) = cut
        ispline1D = ir
    end if
    if (fLJr1D(ia,ir).gt.abs(cut)) then
        fLJr1D(ia,ir) = -cut
    end if
    if (abs(fCr1D(ia,ir)).gt.abs(cut)) then
        fCr1D(ia,ir) = sign(real(1,dp),idealHist1D(ia,4,ir))*cut
    end if
end do
! spline g1D for the solute

```

```

    call spline(histDist,g1D(ia,:),ispline1D,histDistBins,(idealHist1D(ia,2,
ispline1D)+idealHist1D(ia,4,ispline1D)),real(0,dp), &
    & g1D2(ia,:))
    call spline(histDist,fLJr1D(ia,:),ispline1D,histDistBins,idealHist1D(ia,3,
ispline1D),real(0,dp), fLJr1D2(ia,:))
    call spline(histDist,fCr1D(ia,:),ispline1D,histDistBins,idealHist1D(ia,5,
ispline1D),real(0,dp), fCr1D2(ia,:))
    call spline(histDist,emf1D(ia,:),ispline1D,histDistBins,real(0,dp),real(0,dp),
emf1D2(ia,:))
end do ! ia

! note: write out the effective input histogram after averaging/alterations.
write(*,*) 'Writing input histogram after averaging/alterations to "input_hist.out"
...
open(91,file='input_hist.out',status='replace')
write(91,*) '# 1. Distance'
write(91,*) '# 2. g+'
write(91,*) '# 3. fLJ.r+'
write(91,*) '# 4. fC.r+'
write(91,*) '# 5. emf.r+'
write(91,*) '# 6. g-'
write(91,*) '# 7. fLJ.r-'
write(91,*) '# 8. fC.r-'
write(91,*) '# 9. emf.r-'
do ir = 1, histDistBins
    write(91,*) histDist(ir), g1D(1,ir), fLJr1D(1,ir), fCr1D(1,ir), emf1D(1,ir), g1D
(2,ir), fLJr1D(2,ir), fCr1D(2,ir), emf1D(2,ir)
end do
close(91)

! note: write out the ideal histogram after averaging/alterations.
write(*,*) 'Writing ideal histogram after averaging/alterations to "ideal_hist.out"
...
open(92,file='ideal_hist.out',status='replace')
write(92,*) '# 1. Distance'
write(92,*) '# 2. g+'
write(92,*) '# 3. fLJ.r+'
write(92,*) '# 4. fC.r+'
write(92,*) '# 5. g-'
write(92,*) '# 6. fLJ.r-'
write(92,*) '# 7. fC.r-'
do ir = 1, histDistBins
    write(92,*) histDist(ir), -idealHist1D(1,1,ir), idealHist1D(1,2,ir), idealHist1D
(1,4,ir), -idealHist1D(2,1,ir), &
    & idealHist1D(2,2,ir), idealHist1D(2,4,ir)
end do
close(92)

open(93,file='3dhist.out',status='replace')
write(93,*) '# 1. Distance'
write(93,*) '# 2. cosTh'
write(93,*) '# 3. phi'
write(93,*) '# 4. g+'
write(93,*) '# 5. g-'
do ir = 1, histDistBins
    do ith = 1, histCosThBins
        do iphi = 1, histPhiBins
            write(93,*) histDist(ir), histCosTh(ith), histPhi(iphi), g(1,ir,ith,iphi),
g(2,ir,ith,iphi)
        end do
    end do
end do
close(93)
!print*, 'look4', g(1,30,1,1), g(2,30,1,1)

```

```

!debug
write(*,*) 'Writing ideal histogram after averaging/alterations to "ideal_3D.out"
...
open(94,file='ideal_3D.out',status='replace')
write(94,*) '# 1. Distance'
write(94,*) '# 2. cosTheta'
write(94,*) '# 3. Phi'
write(94,*) '# 4. g+'
write(94,*) '# 5. fLJ.r+'
write(94,*) '# 6. fC.r+'
write(94,*) '# 7. g-'
write(94,*) '# 8. fLJ.r-'
write(94,*) '# 9. fC.r-'
do ir = 1, histDistBins
  do ith = 1, histCosThBins
    do iphi = 1, histPhiBins
      write(94,*) histDist(ir), histCosTh(ith), histPhi(iphi), -idealHist(1,1,ir
,ith,iphi), idealHist(1,2,ir,ith,iphi), &
& idealHist(1,8,ir,ith,iphi), -idealHist(2,1,ir,ith,iphi), idealHist
(2,2,ir,ith,iphi), idealHist(2,8,ir,ith,iphi)
    end do
  end do
end do
close(94)

!debug  difference between ideal and measured
!rTmp = int(real(4,dp)/histDistStepSize)
!print*, rTmp, histDistStepSize
!do ith=1,histCosThBins
  !write(55,*) histCosTh(ith), g2D(rTmp,ith), idealhist2D(1,rTmp,ith)
!end do

!debug
!do ir=1,histdistbins
  !write(45,*) histDist(ir), g1D(ir), g1D2(ir), idealHist1D(1,ir), fLJr1D(ir),
idealHist1D(2,ir)
!end do
!do ir=1,100*histdistbins
  !xx = ir*(histDistStepSize/100_dp)
  !call splint(histDist,g1D,g1D2,histDistBins,xx, yy)
  !write(55,*) xx, yy
!end do
!write(65,*) histDist(ispline1D), g1D(ispline1D)

end subroutine spline_hist_array

! read LJ--LJ displacements from file
subroutine R_list
  use cfgData; use ctrlData
  implicit none
  integer :: ios, i
  character(len=16) :: junk
  character(len=128) :: line

  if (c_explicit_R .eq. 'no') then
    explicit_R = .false.
  else if (c_explicit_R .eq. 'yes') then
    explicit_R = .true.

  ios = 0; crdLines = -1
  open(20,file='crd_list.out',status='old')
  do while(ios>=0)
    read(20,'(a)',IOSTAT=ios) line
    crdLines = crdLines + 1
  end do
end subroutine R_list

```

```

end do
close(20)

allocate( explicitDist(crdLines) )

ios = 0
open(20,file='crd_list.out',status='old')
do i = 1, crdLines
  read(20,*,iostat=ios) junk, explicitDist(i)
end do
close(20)
end if

end subroutine R_list

! setup for the average force integral
subroutine setup_compute_avg_force
  use cfgData; use angleData; use ctrlData
  implicit none
  integer :: i
  real(kind=dp) :: psiLF

  write(*,*) "Setting up for average force iteration..."

  if (explicit_R .eqv. .true.) then
    cfgRBins = crdLines
    write(*,*) "Number of R Bins:      ", cfgRBins
  else if (explicit_R .eqv. .false.) then
    cfgRBins = int( (R_max - R_min)/RStepSize + 1 )
    if (cfgRBins .eq. 0) then
      cfgRBins = 1
    end if
    write(*,*) "Number of R Bins:      ", cfgRBins
  end if
  xBins = int( (2 * xz_range)/xzStepSize )
  write(*,*) "Number of X Bins:      ", xBins
  zBins = int( (xz_range)/xzStepSize )
  write(*,*) "Number of Z Bins:      ", zBins

  ! allocate array sizes for axes and average force
  allocate( R_axis(cfgRBins), fAvg(2,cfgRBins), x_axis(xBins), z_axis(zBins) )
  R_axis = 0_dp; fAvg = 0_dp; x_axis = 0_dp; z_axis = 0_dp

  ! allocate arrays for control arrays
  ! frcSPA(solutes, LJ/Coulomb, x, z)
  allocate( frcSPA(2, 2, xBins, zBins), grSPA(xBins, zBins) )

  ! Distance Axes
  do i = 1, cfgRBins
    if (explicit_R .eqv. .true.) then
      R_axis(i) = explicitDist(i)
    else if (explicit_r .eqv. .false.) then
      R_axis(i) = (i-1) * RStepSize + R_min
    end if
  end do
  do i = 1, xBins
    x_axis(i) = (i-1) * xzStepSize - xz_range + xzStepSize/2_dp
  end do
  do i = 1, zBins
    z_axis(i) = (i-1) * xzStepSize + xzStepSize/2_dp
  end do

  ! ANGLES

```

```

allocate( cosThetaLF(cfgCosThBins), sinThetaLF(cfgCosThBins), sinPsiLF(cfgPsiBins),
          cosPsiLF(cfgPsiBins) )

! Theta
! tilt off of z
cfgCosThStepSize = (cosTh_max - cosTh_min) / real(cfgCosThBins, dp)
do i = 1, cfgCosThBins
    cosThetaLF(i) = (i-0.5_dp)*cfgCosThStepSize - cosTh_max
    sinThetaLF(i) = sqrt(abs(1_dp-cosThetaLF(i)**2))
end do
write(*,*) "Config Cos(Theta) Step Size:      ", cfgCosThStepSize

! Psi
! processison about z
cfgPsiStepSize = (psi_max - psi_min) / real(cfgPsiBins, dp)
do i = 1, cfgPsiBins
    psiLF = (i+0.5_dp)*cfgPsiStepSize
    sinPsiLF(i) = sin(psiLF)
    cosPsiLF(i) = cos(psiLF)
end do
write(*,*) "Config Psi Step Size:          ", cfgPsiStepSize

end subroutine setup_compute_avg_force

! do the average force integral
subroutine compute_avg_force
    use cfgData; use histData; use angleData; use ctrlData; use constants; use
    functions
    implicit none
    integer :: r, i, j, ip, omp_get_thread_num !, omp_get_num_threads
    real(kind=dp) :: gx1, gx2, flj, emfVec(3), emf, fx1, fx2, polMeanVec(3), emfVecMag

    write(*,*) "Computing average force..."; flush(6)

    allocate(longRange(cfgRBins)) !debug

    ! Calculate the average force integral for top half of bisecting plane of cylinder
    do r = 1, cfgRBins ! loop lj--lj distances
! do r = cfgRBins, cfgRBins ! loop lj--lj distances !debug
        frcSPA = 0_dp; grSPA = 0_dp
        !$omp PARALLEL DEFAULT( none ) &
        !$omp PRIVATE( ip, i, j, gx1, gx2, fx1, fx2, flj, emfVec, emf, polMeanVec,
        emfVecMag ) &
        !$omp SHARED( nThreads, r, xBins, zBins, cut, R_axis, x_axis, z_axis, histDist,
        histDistBins, g1D, g1D2, fLJr1D, fLJr1D2, &
        !$omp& fCr1D, fCr1D2, emf1D, emf1D2, frcSPA, grSPA, soluteChg, dielectric,
        density, dipole_moment ) &
        !$omp NUM_THREADS( nThreads )
        if ((omp_get_thread_num().eq.0).and.(r.eq.1)) then
            write(*,*) 'Parallel CPUs:      ', nThreads
            !write(*,*) 'Parallel CPUs:      ', omp_get_num_threads()
            flush(6)
        end if
        !$omp DO SCHEDULE( guided )
        do ip = 1, (xBins*zBins)
            ! Convert single index 'ip' to the x and z indicies 'i' and 'j' respectively.
            i = int((ip-1)/zBins)+1 ! x integer
            j = mod(ip-1,zBins)+1 ! z integer

            rSolv1(1) = -R_axis(r)/2_dp - x_axis(i)
            rSolv1(2) = 0_dp
            rSolv1(3) = -z_axis(j)
            rSolv1n(1) = euclid_norm(rSolv1)

```

```

rSolv2(1) = R_axis(r)/2_dp - x_axis(i)
rSolv2(2) = 0_dp
rSolv2(3) = -z_axis(j)
rSolvn(2) = euclid_norm(rSolv2)
!if ((i.eq.350).and.(j.eq.200)) print*, 'x,z,r1,r2', x_axis(i), z_axis(j),
rSolvn !debug 1
!if ((i.eq.350).and.(j.eq.200)) print*, 'r1,r2', rSolv1, rSolv2 !debug 1

if (rSolvn(1).gt.histDist(histDistBins)) then
  gx1 = 0_dp
else
  call splint(histDist,g1D(1,:),g1D2(1,:),histDistBins,rSolvn(1), gx1) !
solute 1 density
end if

if (rSolvn(2).gt.histDist(histDistBins)) then
  gx2 = 0_dp
else
  call splint(histDist,g1D(2,:),g1D2(2,:),histDistBins,rSolvn(2), gx2) !
solute 2 density
end if
gx1 = exp(gx1+gx2)

if ((gx1.gt.1d-6).and.((rSolvn(1).le.histDist(histDistBins)).or.(rSolvn(2).le
.histDist(histDistBins)))) then ! inside vol
  if (rSolvn(1).gt.histDist(histDistBins)) then
    ! outside solute 1 interaction volume use ideal field in CDD
    emfVec = (coulomb_const_kcalmolAq2/dielectric*soluteChg(1)/rSolvn(1)
**2) * (-rSolv1/rSolvn(1))
    fx1 = 0_dp ! LJ force is zero
  else
    call splint(histDist, fLJr1D(1,:), fLJr1D2(1,:), histDistBins, rSolvn(1),
flj) ! lj force
    call splint(histDist, emf1D(1,:), emf1D2(1,:), histDistBins, rSolvn(1), emf
) ! field
    emfVec = emf * rSolv1/rSolvn(1)
    !print*, 'emf:', emf !debug
    fx1 = flj
    !if ((i.eq.350).and.(j.eq.200)) print*, 'emf1', emf !debug 1
  end if

  if (rSolvn(2).gt.histDist(histDistBins)) then ! solute 2
    ! outside solute 2 interaction volume use ideal field in CDD
    emfVec = emfVec + (coulomb_const_kcalmolAq2/dielectric*soluteChg(2)/
rSolvn(2)**2) * (-rSolv2/rSolvn(2))
    fx2 = 0_dp ! LJ force is zero
  else
    call splint(histDist, fLJr1D(2,:), fLJr1D2(2,:), histDistBins, rSolvn(2),
flj)
    call splint(histDist, emf1D(2,:), emf1D2(2,:), histDistBins, rSolvn(2), emf
)

    emfVec = emfVec + emf * rSolv2/rSolvn(2)
    fx2 = flj
    !if ((i.eq.350).and.(j.eq.200)) print*, 'emf2', emf !debug 1
    !if ((i.eq.350).and.(j.eq.200)) print*, 'emf_sum', emfVec !debug 1
  end if

  emfVecMag = euclid_norm(emfVec)
  polMeanVec = (tanh(emfVecMag)**(-1) - (emfVecMag)**(-1))*emfVec/emfVecMag
! Langevin fxn Emf --> <p> @ cell
! Once you have the polarization via Langevin, calculate force
  emfVec = soluteChg(1)*dipole_moment*coulomb_const_kcalmolAq2/rSolvn(1)**3
* &
  & (3*dot_product(rSolv1/rSolvn(1), polMeanVec) * rSolv1/rSolvn(1) -
polMeanVec) ! dipole force solute 1

```



```

        frcSPA(1,2,i,j) = frcSPA(1,2,i,j) - (gx1 * emfVec(1)) ! fd*g.R^hat coulomb
solute 1
        !if ((i.eq.350).and.(j.eq.200)) print*, 'frc1', euclid_norm(emfVec) !debug
1

        emfVec = soluteChg(2)*dipole_moment*coulomb_const_kcalmolAq2/rSolv(2)**3
* &
        & (3*dot_product(rSolv2/rSolv(2),polMeanVec) * rSolv2/rSolv(2) -
polMeanVec) ! dipole force solute 2
        frcSPA(2,2,i,j) = frcSPA(2,2,i,j) + (gx1 * emfVec(1)) ! fd*g.R^hat coulomb
solute 2
        !if ((i.eq.350).and.(j.eq.200)) print*, 'frc2', euclid_norm(emfVec) !debug
1

        frcSPA(1,1,i,j) = frcSPA(1,1,i,j) + (gx1 * fx1 * (-rSolv1(1)/rSolv(1))) !
(f.r)*g.R^hat lj solute 1
        frcSPA(2,1,i,j) = frcSPA(2,1,i,j) + (gx1 * fx2 * ( rSolv2(1)/rSolv(2))) !
(f.r)*g.R^hat lj solute 2
        grSPA(i,j) = grSPA(i,j) + gx1 ! gx1 is the SPA at this point
        end if
    end do !ip
!$omp END DO
!$omp END PARALLEL

! NOTE: Long range correction to mean force for Constant Density Dielectric
medium at large distances
emfVecMag = -soluteChg(1)*soluteChg(2)*coulomb_const_kcalmolAq2*(1-dielectric
**(-1)) * &
        & ( (R_axis(r)*(8*histDist(histDistBins)-3*R_axis(r)))/(24*histDist(
histDistBins)**4) &
        & - log(1+R_axis(r)/histDist(histDistBins))/(8*R_axis(r)**2) &
        & + histDist(histDistBins)/(8*R_axis(r)*(R_axis(r)+histDist(histDistBins))
**2) &
        & - R_axis(r)**2/(32*histDist(histDistBins)**4) &
        & + 3/(16*histDist(histDistBins)**2) )

! Add each cell forces to average and normalize
do i = 1, xBins
    do j = 1, zBins
        fAvg(1,r) = fAvg(1,r) + (frcSPA(1,1,i,j) + frcSPA(2,1,i,j)) * real(0.5,dp)
* z_axis(j) ! lj
        fAvg(2,r) = fAvg(2,r) + (frcSPA(1,2,i,j) + frcSPA(2,2,i,j)) * real(0.5,dp)
* z_axis(j) ! Coulomb
        frcSPA(1,1,i,j) = frcSPA(1,1,i,j)/grSPA(i,j) ! lj solute 1
        frcSPA(2,1,i,j) = frcSPA(2,1,i,j)/grSPA(i,j) ! lj solute 1
        frcSPA(1,2,i,j) = frcSPA(1,2,i,j)/grSPA(i,j) ! coulomb solute 2
        frcSPA(2,2,i,j) = frcSPA(2,2,i,j)/grSPA(i,j) ! coulomb solute 2
    end do !z again
end do !x again
call write_test_out(r) ! write grSPA and frcSPA arrays

! NOTE : After the fact multiply all elements by 2*pi*density/8/pi/pi (2*2pi*pi
/3 (4pi**2)/3 steradians from orientations)
! Number density of chloroform per Angstrom**3 == 0.00750924
fAvg(1,r) = fAvg(1,r)*2*pi*density*xzStepSize*xzStepSize
fAvg(2,r) = fAvg(2,r)*2*pi*density*xzStepSize*xzStepSize !debug+ emfVecMag !
with long range correction added
longRange(r) = emfVecMag !debug
end do !r

!debug
open(95,file='longRange.out',status='replace')
do r = 1, cfgRBins
    write(95,*) R_axis(r), longRange(r)
end do

```

```

close(95)

end subroutine compute_avg_force

! integrate the average force from 'compute_avg_force' to get the PMF.
subroutine integrate_force
  use cfgData; use ctrlData
  implicit none
  integer :: d, f

  allocate( u_dir(2, cfgRBins) )
  u_dir = 0_dp

  do f = 1, 2
    if (explicit_R .eqv. .false.) then
      do d = 1, cfgRBins
        if (d .eq. 1) then
          u_dir(f, cfgRBins) = fAvg(f, cfgRBins) * RStepSize
        else
          u_dir(f, cfgRBins-(d-1)) = u_dir(f, cfgRBins-(d-2)) + fAvg(f, cfgRBins-(d
-1)) * RStepSize
        end if
      end do
    else if (explicit_R .eqv. .true.) then
      do d = 1, cfgRBins
        if (d .eq. 1) then
          u_dir(f, cfgRBins) = fAvg(f, cfgRBins) * (R_axis(cfgRBins)-R_axis(
cfgRBins-1))
          !print*, (R_axis(cfgRBins)-R_axis(cfgRBins-1))
        else
          ! FIXME: is the delta R part of this correct?
          u_dir(f, cfgRBins-(d-1)) = u_dir(f, cfgRBins-(d-2)) + fAvg(f, cfgRBins-(d
-1)) * &
            (R_axis(cfgRBins-(d-1))-R_axis(cfgRBins-d))
          ! it looks like the first value is getting printed twice. Also, the
values might be wrong. Should it be (d-1) and
! (d-0)? instead of -2 and -1?
          !print*, (R_axis(cfgRBins-(d-1))-R_axis(cfgRBins-d))
        end if
      end do
    end if
  end do
end subroutine integrate_force

! write force out and g(r) out to compare against explicit
subroutine write_test_out(r)
  use cfgData; use ctrlData
  implicit none
  integer :: r, i_f, i, j
  character(len=32) :: temp, filename
  character(len=8) :: frmt

  i_f = (r-1) * int(RStepSize*10)
  frmt = '(I3.3)' ! an integer of width 3 with zeroes on the left
  write(temp, frmt) i_f ! converting integer to string using 'internal file'
  filename='hist1D_output.'//trim(temp)//'.dat'

  open(35, file=filename, status='replace')
  write(6,*) "Writing test file: ", filename
  write(35,*) "# 1.  X Distance"
  write(35,*) "# 2.  Z Distance"
  write(35,*) "# 3.  g(r)"

```

```

write(35,*) "# 4. Force.r + LJ"
write(35,*) "# 5. Force.r + Coulomb"
write(35,*) "# 6. Force.r - LJ"
write(35,*) "# 7. Force.r - Coulomb"
write(35,*) "# "
do j = 1, zBins
  do i = 1, xBins
    write(35,898) x_axis(i), z_axis(j), grSPA(i,j), frcSPA(1,1,i,j), frcSPA(1,2,i
    ,j), frcSPA(2,1,i,j), frcSPA(2,2,i,j)
  end do
end do
close(35)

flush(6)

898      format (2(1x,es14.7),5(1x,es14.7))
end subroutine write_test_out

! write output file
subroutine write_output(outFile)
  use cfgData
  implicit none
  character(len=128) :: outFile
  integer :: r

  open(35,file=outFile,status='replace')
  write(6,*) "Writing output file: ", outFile
  write(35,*) "# 1. R Distance"
  write(35,*) "# 2. <f>_LJ"
  write(35,*) "# 3. <f>_Coulomb"
  write(35,*) "# 4. PMF LJ"
  write(35,*) "# 5. PMF Coulomb"
  do r = 1, cfgRBins
    write(35,899) R_axis(r), fAvg(1,r), fAvg(2,r), u_dir(1,r), u_dir(2,r)
  end do
  close(35)

  flush(6)

899      format (5(1x,es14.7)) ! scientific format
end subroutine write_output

```

## 2.9 Fortran Function Subroutines: functions.f90

```

! general fortran functions

module functions
  use prec
  implicit none

contains
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! Euclidian Norm or Magnitude of vector. If fortran compiler is older than fortran
2008 'euclid_norm' replaces 'norm2'
function euclid_norm(v)
  implicit none
  real(kind=dp) :: euclid_norm ! output
  real(kind=dp),intent(in) :: v(:) ! input
  integer :: i

  euclid_norm = 0_dp
  do i = 1, size(v)
    euclid_norm = euclid_norm + v(i)**2
  end do

```

```

    euclid_norm = sqrt(euclid_norm)
end function euclid_norm
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! calculate cross product of two vectors, 'a' and 'b'
function cross_product(a, b)
    implicit none
    real(kind=dp), dimension(3)    :: cross_product    ! output
    real(kind=dp), intent(in)      :: a(:), b(:)      ! inputs not to be changed

    cross_product(1) = a(2)*b(3) - a(3)*b(2)
    cross_product(2) = a(3)*b(1) - a(1)*b(3)
    cross_product(3) = a(1)*b(2) - a(2)*b(1)
end function cross_product
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! rotate 'v' about x in 3D by 'a' radians to give the rotated vector
function rotate_x(a, v)
    implicit none
    real(kind=dp), intent(in)      :: a, v(:)        ! inputs not to be changed
    real(kind=dp), dimension(3)    :: rotate_x       ! output

    rotate_x(1) = v(1)
    rotate_x(2) = cos(a) * v(2) - sin(a) * v(3)
    rotate_x(3) = sin(a) * v(2) + cos(a) * v(3)
end function rotate_x
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! rotate 'v' about y in 3D by 'a' radians to give the rotated vector
function rotate_y(a, v)
    implicit none
    real(kind=dp), intent(in)      :: a, v(:)        ! inputs not to be changed
    real(kind=dp), dimension(3)    :: rotate_y       ! output

    rotate_y(1) = cos(a) * v(1) + sin(a) * v(3)
    rotate_y(2) = v(2)
    rotate_y(3) = -sin(a) * v(1) + cos(a) * v(3)
end function rotate_y
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! rotate 'v' about z in 3D by 'a' radians to give the rotated vector
function rotate_z(a, v)
    implicit none
    real(kind=dp), intent(in)      :: a, v(:)        ! inputs not to be changed
    real(kind=dp), dimension(3)    :: rotate_z       ! output

    rotate_z(1) = cos(a) * v(1) - sin(a) * v(2)
    rotate_z(2) = sin(a) * v(1) + cos(a) * v(2)
    rotate_z(3) = v(3)
end function rotate_z
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! NOTE: the following tridiag, spline, and bicubic interpolation subroutines are
! adopted from Numerical Recipies in FORTRAN 2nd Ed.
! note: all lines marked with the 'xxx' flag are changes to the original NR code.

! Solves for a vector u(1:n) of length n the tridiagonal linear set given by
! equation (2.4.1). a(1:n), b(1:n), c(1:n), and r(1:n)
! are input vectors and are not modified. The parameter NMAX is the maximum
! expected value of n.
subroutine tridiag(a,b,c,r,u,n)
    implicit none
    integer                :: n !, NMAX
    real(kind=dp)          :: a(n), b(n), c(n), r(n), u(n)
!    parameter              :: (NMAX=500)
    integer                :: j
    real(kind=dp)          :: bet, gam(n) !, gam(NMAX) ! One vector of workspace, gam
! is needed.

    if ((b(1).lt.1d-6) .and. (b(1).gt.-1d-6)) then ! ie. it's equal to zero xxx

```

```

! If this happens then you should rewrite your equations as a set of order N
-1, with u2 trivially eliminated.
  write(*,*) 'ERROR: tridag: rewrite equations'
  error stop
end if

bet = b(1)
u(1) = r(1)/bet

! Decomposition and forward substitution.
do j = 2, n
  gam(j) = c(j-1)/bet
  bet = b(j) - a(j)*gam(j)
  if ((bet.lt.1d-6) .and. (bet.gt.-1d-6)) then ! ie. it's equal to zero xxx
    write(*,*) 'ERROR: tridag failed' ! Algorithm fails; see below.
    error stop
  end if
  u(j) = (r(j) - a(j)*u(j-1))/bet
end do

! Backsubstitution.
do j = n-1, 1, -1
  u(j) = u(j) - gam(j+1)*u(j+1)
end do

end subroutine tridag
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! Solves for a vector u(1:n) of length n the tridiagonal linear set given by
equation (2.4.1). a(1:n), b(1:n), c(1:n), and r(1:n)
! are input vectors and are not modified. The parameter NMAX is the maximum
expected value of n.
subroutine symm_tridag(dx,y,n, u)
  implicit none
  integer :: n
  real(kind=dp) :: dx, y(n), u(n)
  ! locally defined
  real(kind=dp) :: a(n), b(n), c(n), r(n)
  integer :: j
  real(kind=dp) :: bet, gam(n)

  ! symmetric system
  a(1) = 0_dp
  a(2:n) = dx/real(6,dp)

  b(1) = 5*dx/real(6,dp)
  b(2:n-1) = 2*dx/real(3,dp)
  b(n) = b(1)

  c(1:n-1) = dx/real(6,dp)
  c(n) = 0_dp

  r(1) = (y(2)-y(1))/dx
  do j = 2, n-1
    r(j) = ((y(j+1)-y(j)) - (y(j)-y(j-1))) / dx
  end do
  r(n) = -(y(n)-y(n-1))/dx

  bet = b(1)
  u(1) = r(1)/bet

  ! Decomposition and forward substitution.
  do j = 2, n
    gam(j) = c(j-1)/bet
    bet = b(j) - a(j)*gam(j)
    if ((bet.lt.1d-6) .and. (bet.gt.-1d-6)) then ! ie. it's equal to zero xxx

```

```

        write(*,*) 'ERROR: symm_tridag failed' ! Algorithm fails; see below.
        error stop
    end if
    u(j) = (r(j) - a(j)*u(j-1))/bet
end do

! Backsubstitution.
do j = n-1, 1, -1
    u(j) = u(j) - gam(j+1)*u(j+1)
end do

end subroutine symm_tridag
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! Given arrays x(1:n) and y(1:n) containing a tabulated function, i.e.,  $y_i = f(x_i)$ 
!, with  $x_1 < x_2 < \dots < x_n$ , and given
! values 'yp1' and 'ypn' for the first derivative of the interpolating function at
! points 1 and n, respectively, this
! routine returns an array y2(1:n) of length n which contains the second
! derivatives of the interpolating function at the
! tabulated points  $x_i$ . If 'yp1' and/or 'ypn' are equal to  $1 \times 10^{30}$  or larger, the
! routine is signaled to set the
! corresponding boundary condition for a natural spline, with zero second
! derivative on that boundary.
subroutine spline( x, y, nmin, n, yp1, ypn, y2 ) !xxx
    implicit none
    integer :: nmin !xxx
    integer :: n
    real(kind=dp),intent(in) :: yp1, ypn, x(:), y(:)
    real(kind=dp),intent(inout) :: y2(:)
    ! Locally defined variables
    integer :: i, k
    real(kind=dp) :: p, qn, sig, un, u(n)

    y2(nmin) = -0.5_dp !xxx
    u(nmin) = ( 3_dp / (x(nmin+1)-x(nmin)) ) * ( (y(nmin+1)-y(nmin)) / (x(nmin+1)-x(
nmin)) - yp1 ) !xxx

    ! this is the decomposition loop of the tridiagonal algorithm. y2 and u are used
    for temporary storage of the decomposed
    ! factors.
    do i = nmin+1, n-1 !xxx
        sig = ( x(i)-x(i-1) ) / ( x(i+1)-x(i-1) )
        p = sig * y2(i-1) + 2
        y2(i) = (sig-1_dp) / p
        u(i) = ( 6_dp * ( (y(i+1)-y(i)) / (x(i+1)-x(i)) - (y(i)-y(i-1)) / (x(i)-x(i
-1)) ) / (x(i+1)-x(i-1)) - sig*u(i-1) ) / p
    end do

    qn = 0.5_dp
    un = ( 3_dp / (x(n)-x(n-1)) ) * ( ypn - (y(n)-y(n-1)) / (x(n)-x(n-1)) )

    y2(n) = ( un - qn * u(n-1) ) / ( qn * y2(n-1) + 1_dp )
    do k = n-1, nmin, -1 ! this is the backsubstitution loop of the tridiagonal
algorithm. !xxx
        y2(k) = y2(k) * y2(k+1) + u(k)
    end do

end subroutine spline
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! spline of a symmetric function like  $f(x)=\cos(x)$  with equal spacing of x values,
! ie. constant dx
subroutine symm_spline( dx, y, nmin, n, y2 ) !xxx dx instead of x-array
    implicit none
    integer :: nmin !xxx
    integer :: n

```

```

real(kind=dp)      :: dx, y(n), y2(n)
! Locally defined variables
integer           :: i, k
real(kind=dp)     :: p, qn, un, u(n)

y2(nmin) = -0.2_dp !xxx symmetric system
u(nmin) = ( 6_dp*(y(nmin+1)-y(nmin)) / (5_dp*dx**2) ) !xxx symmetric system

! this is the decomposition loop of the tridiagonal algorithm. y2 and u are used
for temporary storage of the decomposed
! factors.
do i = nmin+1, n-1 !xxx replaced all 'sig' values with 1/2
  p = y2(i-1)/2_dp + 2
  y2(i) = (-0.5_dp) / p
  u(i) = ( (3_dp * (y(i+1)-2_dp*y(i)+y(i-1))) / dx**2) - u(i-1)/2_dp ) / p
end do

qn = 0.2_dp !xxx symmetric system
un = 6_dp*(y(n-1)-y(n)) / (5_dp*dx**2) !xxx symmetric system

y2(n) = ( un - qn * u(n-1) ) / ( qn * y2(n-1) + 1_dp )
do k = n-1, nmin, -1 ! this is the backsubstitution loop of the tridiagonal
algorithm. !xxx
  y2(k) = y2(k) * y2(k+1) + u(k)
end do

end subroutine symm_spline
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! Given the arrays xa(1:n) and ya(1:n) of length n, which tabulate a function (with
the xa_i's in order), and given the
! array y2a(1:n), which is the output from 'spline' above (2nd derivative of ya),
and given a value of x, this routine returns a
! cubic-spline interpolated value y.
subroutine splint(xa,ya,y2a,n,x, y)
  implicit none
  integer           :: n
  real(kind=dp)    :: xa(:), ya(:), y2a(:), x, y
  ! Locally defined variables
  integer          :: k, khi, klo
  real(kind=dp)    :: a, b, h

  ! We will find the right place in the table by means of bisection. This is
optimal if sequential calls to this routine
  ! are at random values of x. If sequential calls are in order, and closely
spaced, one would do better to store previous
  ! values of klo and khi and test if they remain appropriate on the next call.
  klo = 1
  khi = n
  do
    if (khi-klo .eq. 1) exit ! do while loop with this exit statement instead of
a goto like Numerical Recipes has.
    k = (khi+klo)/2
    if (xa(k) .gt. x) then
      khi = k
    else
      klo = k
    end if
  end do

  ! khi and klo now bracket the input value of x.
  h = xa(khi)-xa(klo)
  if (h .eq. 0_dp) then
    write(*,*) 'ERROR: bad xa input in subroutine: splint' ! the xa's must be
distinct.
    error stop

```

```

    end if
    ! cubic spline polynomial is now evaluated.
    a = (xa(khi)-x)/h
    b = (x-xa(klo))/h
    y = a*ya(klo) + b*ya(khi) + ((a**3-a) * y2a(klo) + (b**3-b) * y2a(khi)) * (h**2)
    /6_dp
end subroutine splint
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
subroutine symm_splint(xa,dx,ya,y2a,n,x, y)
    implicit none
    integer          :: n
    real(kind=dp)    :: xa(n), dx, ya(n), y2a(n), x, y
    ! Locally defined variables
    integer          :: k, khi, klo
    real(kind=dp)    :: a, b

    ! We will find the right place in the table by means of bisection. This is
    optimal if sequential calls to this routine
    ! are at random values of x. If sequential calls are in order, and closely
    spaced, one would do better to store previous
    ! values of klo and khi and test if they remain appropriate on the next call.
    klo = 0
    khi = n+1
    do
        if (khi-klo .eq. 1) exit ! do while loop with this exit statement instead of
a goto like Numerical Recipes has.
        k = (khi+klo)/2
        if (xa(k) .gt. x) then
            khi = k
        else
            klo = k
        end if
    end do

    ! khi and klo now bracket the input value of x.
    ! cubic spline polynomial is now evaluated.
    if (klo.eq.0) then
        a = (xa(khi)-x)/dx
        b = 1-a
        y = a*ya(khi) + b*ya(khi) + ((a**3-a) * y2a(khi) + (b**3-b) * y2a(khi)) * (dx
**2)/6_dp
    elseif (khi.eq.n+1) then
        b = (x-xa(klo))/dx
        a = 1-b
        y = a*ya(klo) + b*ya(klo) + ((a**3-a) * y2a(klo) + (b**3-b) * y2a(klo)) * (dx
**2)/6_dp
    else
        a = (xa(khi)-x)/dx
        b = (x-xa(klo))/dx
        y = a*ya(klo) + b*ya(khi) + ((a**3-a) * y2a(klo) + (b**3-b) * y2a(khi)) * (dx
**2)/6_dp
    end if

end subroutine symm_splint
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! Use the 16 surrounding grid points to take the x1, x2, and cross derivatives
using centered differencing and then calls bcuint.
! However, if all of the grid point values are below a certain cutoff (-10**4) then
call bilinear interpolation instead.
subroutine bicubic_int(cut, x1a,x2a,n1,n2,dx1,dx2,x1min,x2min,ya,x1,x2, ansy)!,
ansy1,ansy2)
    implicit none
    integer          :: n1, n2

```



```

    real(kind=dp)    :: x1a(n1), x2a(n2), ya(4,0:n1+1, 0:n2+1), dx1, dx2, x1min,
x2min, x1, x2      ! input
    ! todo: x1a = histCosTh(:), x2a = histPhi(:), dx1 = histCosThStepSize, dx2 =
histPhiStepSize, ya = gTmp1(ir, :, :),
    ! x1 = cosTh(1), x2 = phi(1), ansy = gx
    ! locally defined
    integer          :: lj, lk, uj, uk
    real(kind=dp), dimension(4) :: y, y1, y2, y12
    real(kind=dp)    :: cut, ansy, ansy1, ansy2

    ! todo: first need to identify what the 4 points surrounding the desired point
are. These are points 1-4 circled in NR p.117
    ! todo: pass this routine the actual variable value for one of the solutes to
get a float_index.
    lj = floor((x1-x1min)/dx1 + 0.5_dp)
    lk = floor((x2-x2min)/dx2 + 0.5_dp)
    uj = lj+1
    uk = lk+1
    ! todo: perhaps after calculating the indicies I could ask if the yvalues are
less than or greater than -10**4 and then either
    ! proceed with the bicubic interp or do bilinear interp.
    ! Interpolation cutoff:
    if ((ya(1,lj,lk).le.cut).and.(ya(1,uj,lk).le.cut).and.(ya(1,lj,uk).le.cut).and.(
ya(1,uj,uk).le.cut)) then
        ! Note: if below the cutoff then bilinear interpolation
        call bilin_interp(x1a,x2a,n1,n2,dx1,dx2,x1min,x2min,ya(1, :, :),x1,x2, ansy)
    else
        ! Note: if above the cutoff then bicubic interpolation
        ! Define points 1-4 from NR. 1 = (lj,lk); 2 = (uj,lk); 3 = (uj,uk); 4 = (lj,
uk). Into 1D arrays as needed by the other subs
        ! and then call bcuint with all the arrays arranged appropriately for it.
        y(1) = ya(1,lj,lk)
        y(2) = ya(1,uj,lk)
        y(3) = ya(1,uj,uk)
        y(4) = ya(1,lj,uk)
        ! gradient along x1
        y1(1) = ya(2,lj,lk)
        y1(2) = ya(2,uj,lk)
        y1(3) = ya(2,uj,uk)
        y1(4) = ya(2,lj,uk)
        ! gradient along x2
        y2(1) = ya(3,lj,lk)
        y2(2) = ya(3,uj,lk)
        y2(3) = ya(3,uj,uk)
        y2(4) = ya(3,lj,uk)
        ! cross derivative
        y12(1) = ya(4,lj,lk)
        y12(2) = ya(4,uj,lk)
        y12(3) = ya(4,uj,uk)
        y12(4) = ya(4,lj,uk)

        ! todo: build in a way of handling cosTh(1) and cosTh(2) and multiplying them
without having to call bicubic_int, (the
        ! current subroutine), twice.
        if (lj.lt.1) lj = 1; uj = lj+1 ! wrap the indicies for the x-axes
        if (uj.gt.n1) uj = n1; lj = uj-1
        if (lk.lt.1) lk = 1; uk = lk+1
        if (uk.gt.n2) uk = n2; lk = uk-1
        call bcuint(y,y1,y2,y12,x1a(lj),x1a(uj),x2a(lk),x2a(uk),x1,x2, ansy,ansy1,
ansy2)
    end if
end subroutine bicubic_int
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

```

```

! Bicubic interpolation within a grid square. Input quantities are y,y1,y2,y12 (as
! described in bcucof); x1l and x1u, the lower
! and upper coordinates of the grid square in the 1- direction; x2l and x2u
! likewise for the 2-direction; and x1,x2, the
! coordinates of the desired point for the interpolation. The interpolated function
! value is returned as ansy, and the
! interpolated gradient values as ansy1 and ansy2. Note: this routine calls bcucof.
subroutine bcuint(y,y1,y2,y12,x1l,x1u,x2l,x2u,x1,x2, ansy,ansy1,ansy2)
  real(kind=dp)    :: ansy,ansy1,ansy2,x1,x1l,x1u,x2,x2l,x2u,y(4),y1(4),y12(4),y2
  (4)
  integer          :: i
  real(kind=dp)    :: t,u,c(4,4)

  call bcucof(y,y1,y2,y12,x1u-x1l,x2u-x2l,c)    ! Get the c's.
  if (x1u.eq.x1l.or.x2u.eq.x2l) then
    write(*,*) "ERROR: bad input in subroutine: bcuint"
  end if
  t=(x1-x1l)/(x1u-x1l)          ! Equation (3.6.4).
  u=(x2-x2l)/(x2u-x2l)
  ansy=0.
  ansy2=0.
  ansy1=0.
  do i=4,1,-1                  ! Equation (3.6.6).
    ansy=t*ansy+((c(i,4)*u+c(i,3))*u+c(i,2))*u+c(i,1)
    ansy2=t*ansy2+(3.*c(i,4)*u+2.*c(i,3))*u+c(i,2)
    ansy1=u*ansy1+(3.*c(4,i)*t+2.*c(3,i))*t+c(2,i)
  end do
  ansy1=ansy1/(x1u-x1l)
  ansy2=ansy2/(x2u-x2l)
  return
end subroutine bcuint
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! Given arrays y,y1,y2, and y12, each of length 4, containing the function,
! gradients, and cross derivative at the four grid
! points of a rectangular grid cell (numbered counterclockwise from the lower left)
! , and given d1 and d2, the length of the grid
! cell in the 1- and 2- directions, this routine returns the table c(1:4,1:4) that
! is used by routine bcuint for bicubic
! interpolation.
subroutine bcucof(y,y1,y2,y12,d1,d2, c)
  implicit none
  real(kind=dp)    :: d1,d2,c(4,4),y(4),y1(4),y12(4),y2(4)
  integer          :: i,j,k,l
  real(kind=dp)    :: d1d2,xx,c1(16),wt(16,16),x(16)
  save             :: wt
  data wt/1,0,-3,2,4*0,-3,0,9,-6,2,0,-6,4,8*0,3,0,-9,6,-2,0,6,-4 &
    & ,10*0,9,-6,2*0,-6,4,2*0,3,-2,6*0,-9,6,2*0,6,-4 &
    & ,4*0,1,0,-3,2,-2,0,6,-4,1,0,-3,2,8*0,-1,0,3,-2,1,0,-3,2 &
    & ,10*0,-3,2,2*0,3,-2,6*0,3,-2,2*0,-6,4,2*0,3,-2 &
    & ,0,1,-2,1,5*0,-3,6,-3,0,2,-4,2,9*0,3,-6,3,0,-2,4,-2 &
    & ,10*0,-3,3,2*0,2,-2,2*0,-1,1,6*0,3,-3,2*0,-2,2 &
    & ,5*0,1,-2,1,0,-2,4,-2,0,1,-2,1,9*0,-1,2,-1,0,1,-2,1 &
    & ,10*0,1,-1,2*0,-1,1,6*0,-1,1,2*0,2,-2,2*0,-1,1/
  d1d2=d1*d2
  do i=1,4                    ! Pack a temporary vector x.
    x(i)=y(i)
    x(i+4)=y1(i)*d1
    x(i+8)=y2(i)*d2
    x(i+12)=y12(i)*d1d2
  end do
  do i=1,16                  ! Matrix multiply by the stored table.
    xx=0.
    do k=1,16
      xx=xx+wt(i,k)*x(k)
    end do
  end do

```

```

        cl(i)=xx
    end do
    l=0
    do i=1,4          ! Unpack the result into the output table.
        do j=1,4
            l=l+1
            c(i,j)=cl(l)
        end do
    end do
    return
end subroutine bcucof
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! bilinearly interpolate
subroutine bilin_interp(x1a,x2a,nx1,nx2,dx1,dx2,x1Min,x2Min,ya,x1,x2, ansy)
    implicit none
    ! input/output data
    integer :: nx1, nx2
    real(kind=dp) :: x1a(nx1), x2a(nx2), dx1, dx2, x1Min, x2Min, ya(0:nx1+1,0:nx2+1)
, x1, x2
    ! locally defined data
    integer :: ix1l, ix1u, ix2l, ix2u
    real(kind=dp) :: f_index, x1l, x1u, x2l, x2u, x1d, x2d, ansy, x1Int, x2Int

    ! x1
    f_index = (x1 - x1Min) / dx1 + 0.5_dp ! take into account half-bin positions
    ix1l = floor(f_index) ! get flanking r indicies
    if (ix1l .ge. nx1) then
        ix1l = nx1
        ix1u = nx1
    else if (ix1l .lt. 1) then
        ix1l = 1
        ix1u = 1
    else
        ix1u = ix1l + 1
    end if
    x1l = x1a(ix1l)
    x1u = x1a(ix1u)
    if ((x1 .lt. x1l) .or. (x1 .gt. x1u)) then
        x1Int = x1l ! when x1 is outside the bounds it gets set to x1l=x1u.
    else
        x1Int = x1
    end if

    ! x2
    f_index = (x2 - x2Min) / dx2 + 0.5_dp ! so index values start at 1
    ix2l = floor(f_index) ! get flanking x2 indicies
    if (ix2l .ge. nx2) then
        ix2l = nx2
        ix2u = nx2
    else if (ix2l .lt. 1) then
        ix2l = 1
        ix2u = 1
    else
        ix2u = ix2l + 1
    end if
    x2l = x2a(ix2l)
    x2u = x2a(ix2u)
    if ((x2 .lt. x2l) .or. (x2 .gt. x2u)) then
        x2Int = x2l
    else
        x2Int = x2
    end if

    ! Note: set fractional distances
    if ( ix1l .eq. ix1u) then ! if x1u=x1l then x1d would become a NaN.

```

```

        x1d = 1_dp
    else
        x1d = (x1Int-x1l)/(x1u-x1l)
    end if
    if ( ix2l .eq. ix2u ) then ! if x2u=x2l then x2d would become a NaN.
        x2d = 1_dp
    else
        x2d = (x2Int-x2l)/(x2u-x2l)
    end if

    ansy = (ya(ix1l,ix2l)*(1-x1d) + ya(ix1u,ix2l)*x1d)*(1-x2d) + (ya(ix1l,ix2u)*(1-
x1d) + &
        & ya(ix1u,ix2u)*x1d)*x2d

end subroutine bilin_interp
end module functions

```