The Solvation Energy of Ions in a Stockmayer Fluid

Cameron John Shock
Michigan Technological University, cjshock@mtu.edu

THE SOLVATION ENERGY OF IONS IN A STOCKMAYER FLUID

By

Cameron J. Shock

A THESIS

Submitted in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

In Physics

MICHIGAN TECHNOLOGICAL UNIVERSITY

2019

This thesis has been approved in partial fulfillment of the requirements for the Degree of MASTER OF SCIENCE in Physics.

Department of Physics

Thesis Advisor:	*Dr. Issei Nakamura*
Committee Member:	*Dr. Ravindra Pandey*
Committee Member:	*Dr. Jacek Borysow*
Committee Member:	*Dr. Patricia Heiden*
Department Chair:	*Dr. Ravindra Pandey*

Dedication

I dedicate this work to all my past teachers. The work you did helped me grow and succeed to get me to this point in my life. Your job is the most important job in the world, and I cannot thank you enough.

Contents

List of Figures

List of Tables

Acknowledgments

I would like to acknowledge my advisor, Dr. Issei Nakamura, for being an excellent advisor and helping me learn to become a better researcher. I would also like to acknowledge my fellow group members and graduate colleagues, as without their support and friendship I would lose my mind. I would like to acknowledge Dr. Mark Stevens and Dr. Amalie Frischknecht, whose assistance has been immensely helpful for this work. And finally I'd like to acknowledge my parents for raising me the way they did so I could find my way through the world and get to where I am today.

Abstract

The solvation of ions in polar solvents has been a long studied system since the early twentieth century. A common technique to calculate the energy associated with ion solvation is the Born Solvation energy equation. This equation assumes an ion is placed in an incompressible, homogeneous dielectric, which is not necessarily representative of a real system. In this work the Stockmayer Fluid Model is used in a molecular dynamics simulation through the software LAMMPS to check the quantitative correctness of the Born equation. It is also shown how solvation energies of ions placed in polymerized and non-polymerized solvents differ. It is shown that solvation energies in non-polymerized solvents are less negative than the predicted Born Solvation energy due to dielectric saturation effects, but are qualitatively similar. Solvation energies of polymerized solvents differ greatly from non-polymerized solvents and the predicted Born Solvation Energy. The reason for this is speculated to be due to compressibility of the solvents or structural and dipolar effects from polymer chains. It is also shown that the Stockmayer Fluid Model can be used to accurately predict experimental results for non-polymeric solvents.

Chapter 1

Introduction

The response of solvent molecules to the introduction of solutes is known as solvation. A simple example would be dissolving salt in water. The nature of these interactions has been studied in depth since the start of the twentieth century. In 1920, Max Born proposed an equation for approximating the solvation energy[7]. Born's equation approximates the change in free energy when moving an ion from a vacuum to a solvent, and relates this energy to ion charge, ion size, and the dielectric constant of the solvent. Nearly a decade later, Debye published his seminal book *Polar Molecules*[8], in which he developed ideas about the microscopic nature of polar gasses and liquids to predict macroscopic properties like the dielectric constant. He also discussed how ions interact with surrounding solvent molecules, introducing the concept of dielectric saturation where in the presence of high electric fields, solvent dipoles reorient themselves in a way that results in a lowering of the dielectric constant. Later, Lars Onsager read Debye's book and noticed an error in his derivation for the dielectric constant, publishing a correction himself[9]. F. Booth then utilized this corrected theory to derive a new equation for the dielectric saturation effect[10]. Also around this time, in the mid-twentieth century, scientists were understanding how solvent complexes formed around ions, indicating solvation is a local effect in contrast to Born's original proposal[11]. In Chapter 2 these theories are discussed in more detail.

Today, with access to powerful supercomputers we can use computational techniques to test these theories and probe aspects of solvation and dielectric effects we otherwise could not. One such technique is molecular dynamics, where movements of molecules or coarse-grained particles are simulated to predict kinetic and thermodynamic properties of systems. With this technique we can test our models to see how it compares to our theories and experiment. This is discussed in Chapter 3.

This work focuses on the most basic of systems, the solvation of an individual ion into a dipolar solvent. A series of molecular dynamics simulations, through the program LAMMPS developed by Sandia National Laboratories[12], were employed to gain insight on the nature of these interactions. The system is modeled using a Stockmayer Fluid Model, wherein all molecules are treated as simple Lennard-Jones spheres with point dipole moments and point charges[13]. The simulation methods are detailed in Chapter 4.

We investigate how predictions made with the Born Solvation energy equation compare with this model. We also look into how solvation energy is affected by the polymerization of solvent molecules. Finally, preliminary results are given of work done to determine the usefulness of the Stockmayer Fluid Model for predicting experimental results. The results of this work are discussed in Chapter 5.

Chapter 2

Theory of Solvation

2.1 Born Solvation Energy

The first equation for calculating solvation energy was introduced by Max Born[7]. Essentially the solvation energy is an estimation of the change in free energy when moving an ion from a vacuum to a solvent with a continuous bulk dielectric constant, ϵ_r. It can be derived as follows. First we can calculate the electrostatic energy of a single ion of radius a in a vacuum.

$$U = \frac{\epsilon_0}{2} \int_{outside} E^2 dV \tag{2.1}$$

The electric field for an ion is given by

$$E = \frac{q}{4\pi\epsilon_0 r^2} \tag{2.2}$$

so our energy becomes

$$U = \frac{q^2}{32\pi^2\epsilon_0}\int_{outside}\frac{1}{r^2}dV$$

$$= \frac{q^2}{32\pi^2\epsilon_0}4\pi\int_a^\infty\frac{1}{r^2}dr$$

Thus we find the energy of an ion in vacuum to be

$$U = \frac{q^2}{8\pi\epsilon_0 a} \tag{2.3}$$

Similarly we can do the same for an ion in a solvent with a bulk dielectric constant to find

$$U = \frac{q^2}{8\pi\epsilon_0\epsilon_r a} \tag{2.4}$$

Subtracting equations 2.4 and 2.3 gives us the change in energy in moving the ion from the vacuum to the solvent, giving us the Born Solvation Energy for a single ion.

$$\Delta G = -\frac{q^2}{8\pi\epsilon_0 a}(1 - \frac{1}{\epsilon_r}) \tag{2.5}$$

The variables to note here are the ionic charge q, ionic radius a, and dielectric constant ϵ_r. The charge is limited to the valency of the ion, such as monovalent, divalent, trivalent, etc. Ionic radius is determined by choice of ionic molecule. The most interesting aspects of this equation come from the dielectric constant of the medium. This is determined by the choice of solvent and depends on number density, dipole moment of the solvent molecules, and temperature. This will be discussed in more detail in the next section.

The Born Solvation Energy continues to be used by scientists to this day. Some have shown it to be in good agreement with experimental data[14], others have found it to be quite questionable[6]. One of the major aspects of this work has been to determine the qualitative and quantitative validity of this equation, much of which will be discussed in Chapter 5.

2.2 Onsager Theory

The dielectric constant is a macroscopic property of a material resulting from microscopic electronic interactions. A simple situation in which we observe a dielectric constant is that of a parallel plate capacitor. The formulation given in this section has mainly been worked out from Raju[15] based on the theories of Debye[8] and Onsager[9], but has been reorganized to fit the needs of this work.

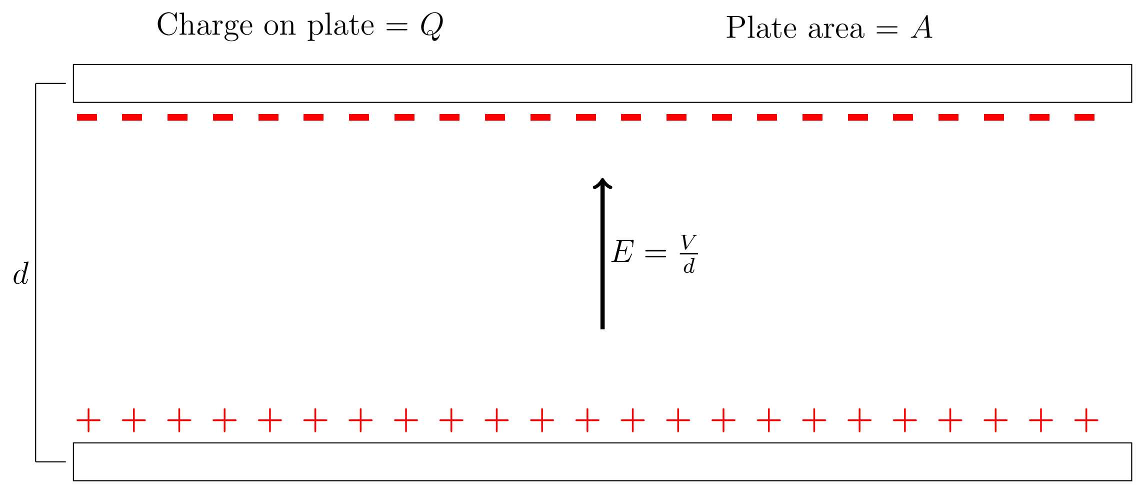


Figure 2.1: Diagram of a parallel plate capacitor described in this section.

Let's take a capacitor in a vacuum. Say we have two electrodes both with area A spaced a distance d apart. When a potential difference of V exists between the plates, the magnitude of the field at any point between them is $E = V/d$. We know that the capacitance is proportional to the area of the plates divided by the distance, the constant of proportionality being the vacuum permittivity ϵ_0.

$$C_0 = \frac{\epsilon_0 A}{d} \tag{2.6}$$

We also know that the capacitance is the charge on the plates divided by the potential difference, thus we can find the charge on each plate is

$$Q_0 = A\epsilon_0 E \tag{2.7}$$

Now if we instead place a dielectric material between the two electrodes, replacing the vacuum, we obtain

$$Q = A\epsilon_0\epsilon_r E \tag{2.8}$$

meaning the charge on the plates is proportional to the charge in the case of a vacuum but times some dielectric constant ϵ_r. The dielectric constant is always greater than unity so the amount of charge on the plates has increased. This is due to the build up of bound charges on the dielectric surface.

Now a dipole is defined as a pair of opposite charges with charge q separated by some distance d, and knowing this we can define a quantity known as the dipole moment,

$$\mu = qd \tag{2.9}$$

In the case of the parallel plate capacitor, if we take the charge on the plates to be the difference in charge from the vacuum to that introduced by adding the dielectric we can obtain its dipole moment as

$$\mu = AE\epsilon_0(\epsilon_r - 1)d \tag{2.10}$$

Now we'll also define a quantity known as the polarization P, which is the dipole moment per unit volume. In the case of the capacitor we have

$$P = \frac{\mu}{Ad} = E\epsilon_0(\epsilon_r - 1) \tag{2.11}$$

Polarization in dielectrics results from various mechanisms. In particular we are going to look at electronic polarization and orientational polarization, which are important for understanding Onsager's Theory for dielectrics. Electronic polarization comes from the introduction of an external field and how the charge distributions are affected. Orientational polarization arises from the change in direction of polar molecules in response to an external field.

2.2.1 Electronic Polarization

To discuss electronic polarization let us first start with a spherically symmetric atom with radius R. If we introduce an external electric field then a dipole moment can be induced

$$\mu_e = (4\pi\epsilon_0 R^3)E \tag{2.12}$$

What may be noticed here is that the term in the parentheses is constant, so the induced dipole moment is proportional to the external electric field. From here we can define yet another term, the electronic polarizability α_e, which is the dipole moment induced per electric field strength. So in a dielectric, the polarizability can be shown to be

$$\vec{P} = N\alpha_e\vec{E} \tag{2.13}$$

where N is the number of atoms per unit volume. Now relating this with equation

2.11 we obtain

$$N\alpha_e\vec{E} = \vec{E}\epsilon_0(\epsilon_r - 1)$$

$$\epsilon_r = \frac{N\alpha_e}{\epsilon_0} + 1$$

Now replacing α_e with μ_e/E using μ_e we found in equation 2.12 we obtain

$$\epsilon_r = 4\pi NR^3 + 1 \tag{2.14}$$

So here we have the beginnings of formulating the dielectric constant in terms of the number density N and the radius of a particle R. However, this assumes that the neighboring molecules do not influence the polarization, which is quite the assumption to make! Thus we need to take this into account to formulate a more encompassing theory.

For this we need to introduce the concept of the internal field E_i. This is the electric field experienced by a molecule from the polarization of the surrounding molecules. Using this idea, the dipole moment of a molecule induced by electronic polarization becomes

$$\mu_e = \alpha_e \vec{E}_i \tag{2.15}$$

To calculate the internal field we can use the following technique. We will assume our molecule lies inside a spherical cavity, so we wish to calculate the field at the center. To do this we need to add up all of the fields that contribute to this field. These fields are: $\vec{E}_1$, the electric field from the free charges on the electrodes of the capacitor; $\vec{E}_2$, the field from the bound charges of the dielectric; $\vec{E}_3$, the field from the charges on the inner wall of the cavity; $\vec{E}_4$, the field from the atoms inside the cavity. The internal field is then

$$\vec{E}_i = \vec{E}_1 + \vec{E}_2 + \vec{E}_3 + \vec{E}_4 \tag{2.16}$$

Firstly, what one can notice is that fields $\vec{E}_1$ and $\vec{E}_2$ come from outside the cavity, thus they are simply the external field.

$$\vec{E} = \vec{E}_1 + \vec{E}_2 \tag{2.17}$$

To calculate $\vec{E}_3$ we will consider a small area dA on the surface of the cavity. We also have an angle θ which is the angle between the directions of the external field $\vec{E}$ and the normal component of the polarization $\vec{P}_n$.

$$\vec{P}_n = \vec{P}\cos\theta \tag{2.18}$$

So the charge on the small area dA is

$$dq = P\cos\theta dA \tag{2.19}$$

We then find the electric field at the center of the cavity due to the charge dq is

$$d\vec{E}_3' = \frac{\vec{P}\cos\theta}{4\pi\epsilon_0 r^2}dA \tag{2.20}$$

But we want the part of the field parallel to the applied external field. Thus

$$d\vec{E}_3 = d\vec{E}_3'\cos\theta = \frac{\vec{P}\cos^2\theta}{4\pi\epsilon_0 r^2}dA \tag{2.21}$$

Now we note that all the surface elements that make an angle θ with $\vec{E}$ have some $d\vec{E}_3$. The area of this being

$$dA = 2\pi r^2 \sin\theta d\theta \tag{2.22}$$

The total area of this gives a ring such that

$$d\vec{E}_3 = \frac{\vec{P}\cos^2\theta}{4\pi\epsilon_0 r^2} * 2\pi r^2 \sin\theta d\theta \tag{2.23}$$

Thus we can find the total to be

$$\vec{E}_3 = \int_0^\pi \frac{\vec{P}\cos^2\theta}{2\epsilon_0}\sin\theta d\theta = \frac{\vec{P}}{3\epsilon_0} \tag{2.24}$$

For $\vec{E}_4$, we find that due to symmetry of the dipole moments inside the cavity that $\vec{E}_4 = 0$. Thus the internal field is found to be

$$\vec{E}_i = \vec{E} + \frac{\vec{P}}{3\epsilon_0} \tag{2.25}$$

This is also known as the Lorentz Field. Now if we substitute from equation 2.11 we obtain

$$\vec{E}_i = \vec{E}\left(\frac{2+\epsilon_r}{3}\right) \tag{2.26}$$

Now combining equations 2.13 and 2.11 and doing some algebra we can come up with

$$\frac{\epsilon_r - 1}{\epsilon_r + 2} = \frac{N_A \alpha_e}{3\epsilon_0 V} \tag{2.27}$$

Here N_A is Avagadro's number and V is molar volume. The relation is as follows

$$N_A = \frac{N \times M}{\rho} = N \times V \tag{2.28}$$

where M is the number of moles and ρ is the density. Substituting this into equation 2.27 we obtain

$$\frac{\epsilon_r - 1}{\epsilon_r + 2}\frac{M}{\rho} = \frac{N_A \alpha_e}{3\epsilon_0} = R \tag{2.29}$$

This equation is known as the Claussius-Mossotti equation. R is known as the molar polarizability. This equation is only applicable to small densities because we made the assumption that neighboring molecules do not influence polarizability.

One other thing to mention is the relation found by Maxwell that $\epsilon_r = n^2$ so using this we can come up with the Lorentz-Lorenz equation.

$$\frac{\epsilon_r - 1}{\epsilon_r + 2} = \frac{n^2 - 1}{n^2 + 2} \tag{2.30}$$

2.2.2 Orientational Polarization

In the previous section we assumed our cavity surrounded a non-polar molecule. But now let us consider the case of a molecule with some permanent dipole moment. In this case the polarization in the absence of an applied electric field is zero because the molecules will rotate in random directions giving no preferred angle. However, when an electric field is applied the number of dipoles confined to a solid angle $d\Omega$ is

$$n(\theta) = Ae^{-\nu/k_B T} d\Omega \tag{2.31}$$

where ν is the potential energy of the dipole and A is a constant that depends on the number of dipoles. Next, we know the surface area on the cavity sphere between angles θ and $\theta + d\theta$ is given by equation 2.22 and the solid angle is defined as dA/r^2 so between those angles

$$d\Omega = 2\pi \sin\theta d\theta \tag{2.32}$$

Now we note that the potential energy of a dipole is

$$\nu = -\mu E \tag{2.33}$$

but since they're at angle θ they are reduced to

$$\nu = -\mu E \cos\theta \tag{2.34}$$

So we can rewrite equation 2.31 as

$$n(\theta) = Ae^{-\mu E \cos\theta / k_B T} * 2\pi \sin\theta d\theta \tag{2.35}$$

What we can then find is that the contribution to the dipole moment by all the dipoles confined to $d\Omega$ is

$$\mu(\theta) = n(\theta)\mu \cos\theta \tag{2.36}$$

From this we can find the average moment per dipole in the direction of the applied field. To do so we take the dipole moment of all the dipoles divided by all of the dipoles.

$$\begin{aligned}\mu_0 &= \frac{\int_0^\pi n(\theta)\mu\cos\theta d\theta}{\int_0^\pi n(\theta)d\theta} \\ &= \frac{\int_0^\pi Ae^{\mu E\cos\theta/k_BT} * 2\pi\sin\theta * \mu\cos\theta d\theta}{\int_0^\pi Ae^{\mu E\cos\theta/k_BT} * 2\pi\sin\theta d\theta}\end{aligned} \tag{2.37}$$

To simplify solving this we'll take the following substitutions

$$x = \frac{\mu E}{k_BT}$$

$$y = \cos\theta$$

to obtain

$$\frac{\mu_0}{\mu} = \frac{\int_1^{-1} ye^{xy}dy}{\int_1^{-1} e^{xy}dy} \tag{2.38}$$

Solving the integrals and simplifying gives us the following

$$\frac{\mu_0}{\mu} = \coth x - \frac{1}{x} = L(x) \tag{2.39}$$

which is known as the Langevin function. Now taking the Taylor Series expansion we find

$$L(x) = \coth x - \frac{1}{x} = \frac{1}{x}\left(\frac{x^2}{3} - \frac{x^4}{45} + ...\right) \tag{2.40}$$

So in the case of low electric fields we can use the following approximation

$$L(x) \approx \frac{x}{3} = \frac{\mu E}{3k_B T} \tag{2.41}$$

Thus we find

$$\mu_0 = \frac{\mu^2 E}{3k_B T} \tag{2.42}$$

giving us the following polarizability

$$\alpha_0 = \frac{\mu_0}{E} = \frac{\mu^2}{3k_B T} \tag{2.43}$$

So from this we find the polarization due to the orientation of the dipoles to be

$$\vec{P} = N\alpha_0\vec{E} = \frac{N\mu^2\vec{E}}{3k_BT} \tag{2.44}$$

The total polarization from both the electronic and orientational polarizations is then

$$\vec{P} = N\vec{E}\left(\alpha_e + \frac{\mu^2}{3k_BT}\right) \tag{2.45}$$

The previous formulation was based on the work done by Debye, known best for his extensive work on polar molecules. Now while this formulation takes into account the orientational aspects of the dipoles, it only works well for low polar gasses. It fails for polar liquids because it does not take into account a change in the electric field known as the reaction field.

2.2.3 Reaction Field

By having a system in which a dipole is encased in a spherical cavity we must take into account how the dipole interacts with itself. To do this we invoke the concept of a reaction field.

The reaction field is essentially an increase of the field in the cavity. To calculate it we need to look at the boundary conditions of our system. We will consider ϕ_1 to be the potential in the dielectric outside the spherical cavity, and ϕ_2 to be the potential inside the spherical cavity. Since we know the dipoles influence dies down as we move away from it we can say our first boundary condition is $(\phi_1)_{r\rightarrow\infty} = 0$.

Also we'll take note that from the Laplace Equation

$$\nabla^2\phi = 0 \tag{2.46}$$

the general solution comes out to be

$$\phi = \left(Ar + \frac{B}{r^2}\right)\cos\theta \tag{2.47}$$

so we will say for outside and inside the cavity we have

$$\phi_1 = \left(A_1 r + \frac{B_1}{r^2}\right)\cos\theta \tag{2.48}$$

$$\phi_2 = \left(A_2 r + \frac{B_2}{r^2}\right)\cos\theta \tag{2.49}$$

However, due to our first boundary condition we find that $A_1 = 0$.

The next boundary condition we must take into account is that the potentials on the surface of the sphere from inside and outside must be the same, $(\phi_1)_{r=R} = (\phi_2)_{r=R}$. Using this boundary condition with equations 2.48 and 2.49 we obtain

$$B_1 = A_2 R^3 + B_2 \tag{2.50}$$

The next boundary condition also deals with the surface of the cavity, the normal component of the flux density across it is continuous, $\left(\frac{\partial \phi_2}{\partial r}\right)_{r=R} = \epsilon_r \left(\frac{\partial \phi_1}{\partial r}\right)_{r=R}$. From this boundary condition we can find that

$$A_2 = \frac{2}{R^3}(B_2 - B_1 \epsilon_r) \tag{2.51}$$

Our final boundary condition is if the boundary is moved very far away from the center, $R \to \infty$, then the potential inside is given by

$$(\phi_2)_{R\to\infty} = \frac{\mu \cos\theta}{4\pi\epsilon_0 r^2} \tag{2.52}$$

From here we see that $B_2 = \frac{\mu}{4\pi\epsilon_0}$. So now we can plug what we've found for A_1 and B_2 into equations 2.50 and 2.51 to find

$$A_2 = \frac{2\mu}{4\pi\epsilon_0 R^3}\frac{(1-\epsilon_r)}{(1+2\epsilon_r)}$$

$$B_1 = \frac{\mu}{4\pi\epsilon_0}\left(\frac{3}{1+2\epsilon_r}\right)$$

Thus we find the potentials to be

$$\phi_1 = \frac{\mu\cos\theta}{4\pi\epsilon_0 r^2}\left(\frac{3}{2\epsilon_r+1}\right) \tag{2.53}$$

$$\phi_2 = \frac{\mu\cos\theta}{4\pi\epsilon_0}\left[\frac{1}{r^2} + \frac{2r(1-\epsilon_r)}{R^3(2\epsilon_r+1)}\right] \tag{2.54}$$

Now we will define ϕ_r to be the potential at some position r due to a dipole in a vacuum, $\phi_r = \frac{\mu\cos\theta}{4\pi\epsilon_0 r^2}$. Using this, we find the change in potential by the introduction of our spherical cavity to be

$$\Delta\phi_1 = \phi_1 - \phi_r = \frac{2(1-\epsilon_r)}{(2\epsilon_r+1)}\frac{\mu\cos\theta}{4\pi\epsilon_0 r^2} \tag{2.55}$$

$$\Delta\phi_2 = \phi_2 - \phi_r = \frac{2(1-\epsilon_r)}{(2\epsilon_r+1)}\frac{\mu r\cos\theta}{4\pi\epsilon_0 R^3} \tag{2.56}$$

$\Delta\phi_2$ shows that there is an increase of the field in the cavity. This increase is the reaction field.

$$\vec{R} = \frac{2(\epsilon_r - 1)\mu}{4\pi\epsilon_0 R^3(2\epsilon_r + 1)} \tag{2.57}$$

2.2.4 Onsager Equation

In Debye's formulation for orientational polarization he neglected the effect from the reaction field. Onsager claimed that the field that acts on the molecule in the cavity is made up of the cavity field and the reaction field. The reaction field does not cause torque on the molecule so the electric field calculated by Onsager is lower than that of Debye by an amount equal to the reaction field. So in this formulation our spherical cavity contains a dipole moment μ at the center and has the same radius as our molecule.

First we must calculate the field, which consists of the external electric field in the z direction, the field from the polarization of the dielectric material, and the reaction field from the molecular dipole itself. When combined together we obtain the Lorentz field $\vec{E}_i$.

$$\vec{E_i} = \vec{E}\left(\frac{2+\epsilon_r}{3}\right) \tag{2.58}$$

Now we will make the assumption that the dielectric constant inside the cavity is $\epsilon_2 = 1$, so reaction field takes the form of the equation 2.57. From that equation we'll also define a constant value to make the calculation easier.

$$f = \frac{2(\epsilon_r - 1)}{4\pi\epsilon_0 r^3(2\epsilon_r + 1)} \tag{2.59}$$

The previous calculations assumed that the dipole moment was constant on a rigid molecule, but the reaction field increases the dipole moment by αR, so we'll denote the new increased reaction field as R_m.

$$\begin{aligned}\vec{R_m} &= f(\vec{\mu} + \alpha\vec{R_m}) \\ &= \frac{f\vec{\mu}}{1 - f\vec{\mu}} \\ &= \frac{2(\epsilon_r - 1)\vec{\mu}}{4\pi\epsilon_0 r^3(2\epsilon_r + 1) - 2\vec{\mu}(\epsilon_r - 1)}\end{aligned} \tag{2.60}$$

Next we will take the Claussius-Mossotti equation 2.29, equation 2.28, and we'll use Maxwell's relation to replace $\epsilon_r = n^2$ to solve for α as

$$\alpha = \frac{3\epsilon_0}{N}\left(\frac{n^2-1}{n^2+2}\right) \tag{2.61}$$

Also note that $\frac{4}{3}\pi N r^3 = 1$. Substituting these into equation 2.60 we find

$$\vec{R_m} = \frac{2N(\epsilon_r - 1)(n^2+2)\vec{\mu}}{9\epsilon_0(n^2+2\epsilon_r)} \tag{2.62}$$

Thus we obtain the modified dipole moment to be

$$\begin{aligned} \mu_m &= \mu + \alpha R_m \\ &= \frac{\mu}{1-f\alpha} = \frac{2\epsilon_r + 1}{3(2\epsilon_r + n^2)}(n^2+2)\mu \end{aligned} \tag{2.63}$$

Now when directed by an external electric field the average value of the reaction field in the direction of $\vec{E}$ is $\vec{R_m}\langle\cos\theta\rangle$, assuming the reaction field follows the molecular dipole moment μ instantaneously. Since $\vec{R_m}$ is always in the same direction as $\vec{\mu}$, $\vec{R_m}\langle\cos\theta\rangle$ does not contribute to the torque. So then the contribution to the internal field is

$$E = E_i - R_m\langle\cos\theta\rangle \tag{2.64}$$

We already know from equations 2.38 and 2.41 that at low field strengths we have

$$\langle \cos\theta \rangle = \frac{\mu_0}{\mu} = \frac{\mu E}{3k_B T} \tag{2.65}$$

from which we can see

$$E = \frac{3k_B T \langle \cos\theta \rangle}{\mu} \tag{2.66}$$

which when plugged into equation 2.64 gives

$$\frac{3k_B T \langle \cos\theta \rangle}{\mu} = E_i - R_m \langle \cos\theta \rangle \tag{2.67}$$

which can be reorganized to obtain

$$\langle \cos\theta \rangle = \frac{\mu E_i}{\mu R_m + 3k_B T} \tag{2.68}$$

Thus we can see that

$$\mu_0 = \frac{\mu^2 E_i}{\mu R_m + 3k_B T} \tag{2.69}$$

and since we know $\alpha_0 = \mu_0/E_i$ we find that the orientational polarizability when taking the reaction field into account is

$$\alpha_0 = \frac{\mu^2}{\mu R_m + 3k_B T} \tag{2.70}$$

So the total polarization is then

$$P = NE(\alpha_e + \alpha_0) = NE\left(\alpha_e + \frac{\mu^2}{\mu R_m + 3k_B T}\right) \tag{2.71}$$

Now after substituting equation 2.62 into this and some lengthy algebra (detailed in Appendix A) we finally obtain the Onsager Equation.

$$\frac{(\epsilon_r - n^2)(2\epsilon_r + n^2)}{\epsilon_r(n^2+2)^2} = \frac{N\mu^2}{9\epsilon_0 k_B T} \tag{2.72}$$

With this we can use just the number density and permanent dipole moment of our solvent molecules and the temperature of the system to predict the dielectric constant of the solvent. This equation was used in this work to predict the dielectric constant for use in the Born Solvation Energy equation. Hopefully going through this derivation can help us better understand the nature of ion solvation in polar liquids.

2.3 Dielectric Saturation

When an ion is introduced to a solvent the local solvent dipoles reorient themselves in response to its electric field. This results in an overall lowering of the local dielectric constant around the ion. This effect is known as dielectric saturation.

The idea was first published by Debye in 1929[8], but as we saw in the previous section Debye's formulation for the dielectric constant needed some corrections by Onsager. Onsager never published anything on the effect of dielectric saturation himself but F. Booth was able to successfully use Onsager's method to describe it, as well as Kirkwood's method (not utilized for this work)[10].

Booth formulated the idea as follows. First we note that due to electronic polarization in some field E our dielectric constant is given by

$$\epsilon_r = 1 + \frac{4\pi \bar{M}_E}{VE} \tag{2.73}$$

where $\bar{M}_E$ is the mean dipole moment of some volume V due to the dipoles induced by the external field. This mean dipole moment can be written as

$$\bar{M}_E = N\mu\langle\cos\theta\rangle \tag{2.74}$$

and we know from equations 2.65 and 2.38 that in the high field limit we obtain

$$\langle\cos\theta\rangle = L\left(\frac{3E\mu}{2k_BT}\right) \tag{2.75}$$

where $L(x)$ is the Langevin Function given in equation 2.39. Now if we take the case of just a single molecule then

$$\epsilon_r = 1 + \frac{4\pi N_0\mu}{E}L\left(\frac{3\mu E}{2k_BT}\right) \tag{2.76}$$

where N_0 is the number of molecules per unit volume. This is known as the Booth Equation. It is known that the saturation effect becomes important for electric fields greater than 10^7 volts per cm[8, 10, 16]. When plotted for a single ion as a function of distance you can see the effect, as shown in Figure 2.2.

Figure 2.3 is from a paper by Gong and Freed[1] showing their calculated dielectric constant as a function of the distance between two charges, plotted for like and unlike

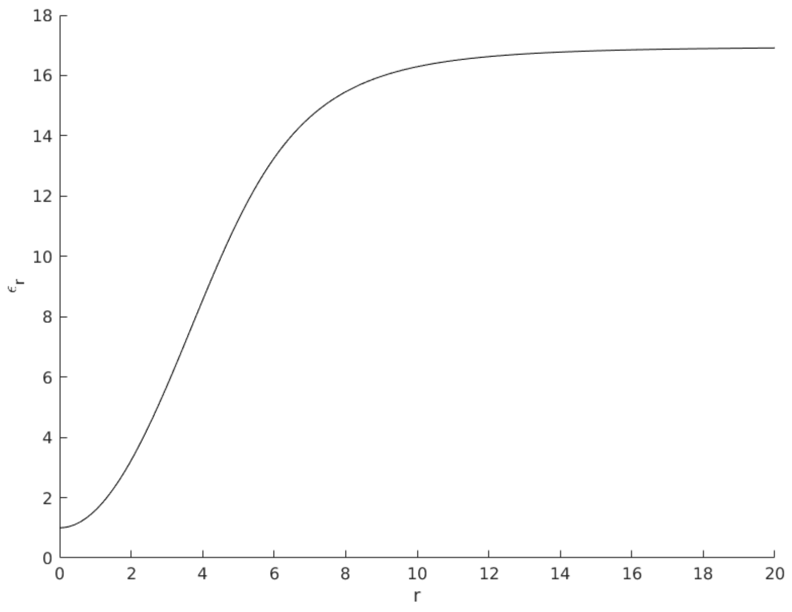


Figure 2.2: Plot of Booth equation for an arbitrary charge as a function of an arbitrary distance.

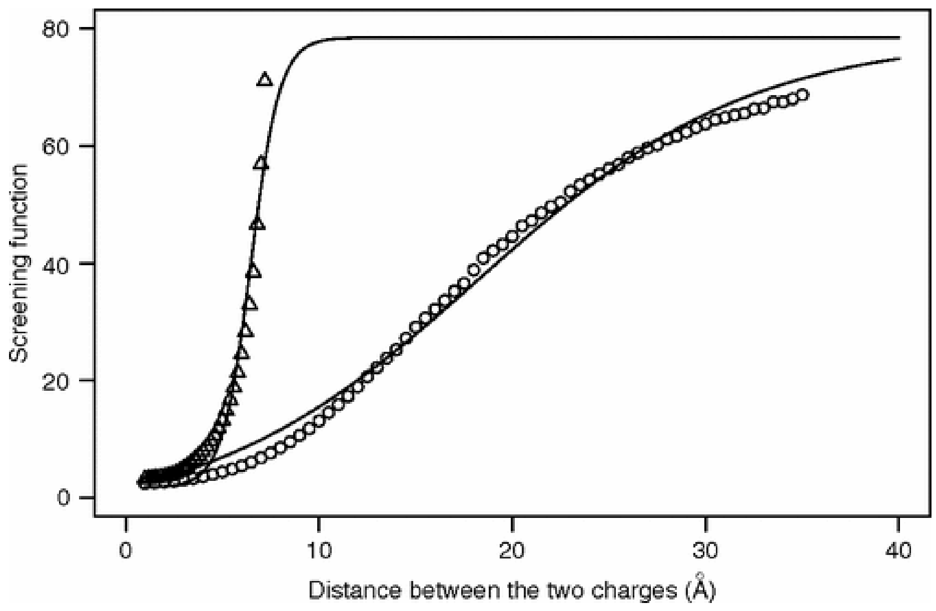


Figure 2.3: Plot of dielectric constant between two charges. The triangle line corresponds to dislike charges and the circle line corresponds to like charges. Retrieved from Ref.[1]

charges. This displays the importance of taking into account the dielectric saturation effect when discussing the solvation of ions (Note how Born Solvation Energy doesn't utilize this).

2.4 Local Structure

In the previous sections we utilized the idea of spherical cavities surrounding our molecules and ions. But how is this idea justified? It comes about from the structure formed around the ion by the surrounding solvent molecules. As is shown in Figure 2.4 a series of complexes surrounds the ion in the local vicinity.

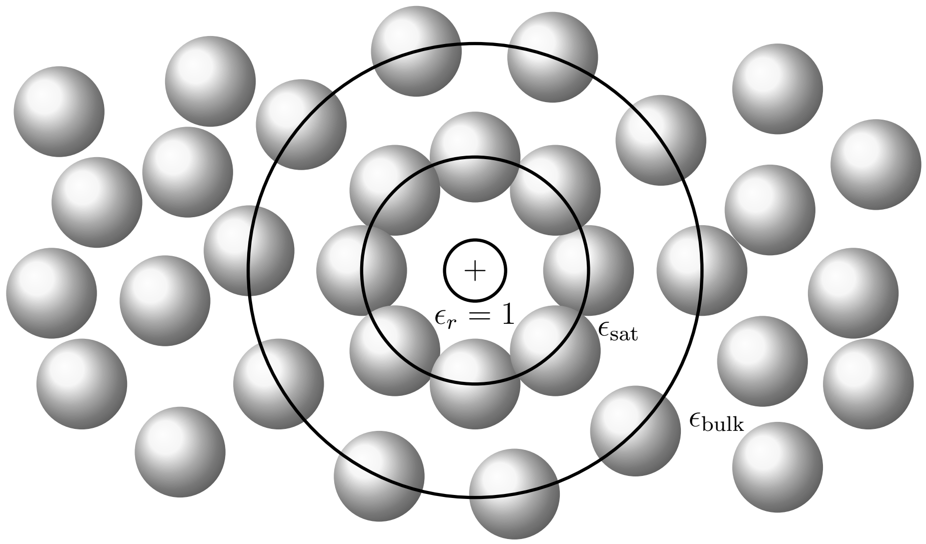


Figure 2.4: Diagram showing solvation shell structures around an ion.

Solvation shells have been discussed since the mid twentieth century, particularly hydration shells involved with aqueous electrolytic solutions[11]. We can consider an ion contained in a spherical cavity with some dielectric constant $\epsilon_r = 1$. Outside is a monolayer of solvent molecules surrounding this cavity in a state where dielectric saturation is reached. Outside of that is another layer of solvent molecules, lesser in density, compressed due to electrostriction. This can be modeled as multiple layers depending on the strength of the ion's electric field (based on its valency). Outside these layers we can consider the solvent as a bulk dielectric material[17].

Because of this structure and the effects of charge screening, only the local effects around the ion are important to consider in ion solvation. It should be noted that the ideas of these last several sections counter that of the Born Solvation Energy introduced at the start of the chapter. Since only the local structure matters, the bulk dielectric constant, ϵ_r, is not sufficient in fully describing the dielectric effects on the free energy. Current approaches to calculating solvation energy involve self-consistent mean field theories, but the goal of this work is to show that even a simple computational model may be enough to accurately describe ion solvation.

Chapter 3

Molecular Dynamics

Molecular dynamics(MD) is a computational technique used to model the mechanics and thermodynamics of particles on the atomic and mesoscopic scales. Simulations can be atomistic, meaning all atoms of a molecule are explicitly modeled, or coarse-grained, where sets of atoms or molecules can be grouped up and treated as a single particle. Classical MD integrates over Newton's equations of motion to simulate the dynamics of molecular systems. The following sections will go into detail about how this is done, other techniques required to prevent errors, and what software can be used.

3.1 Verlet Algorithm

The Verlet Algorithm is a kind of leap-frog method for integrating Newton's equations of motion. It was originally conceived by Delambre and later rediscovered by Loup Verlet. It has a simple derivation and is preferable to most other integration methods. This is because the computational cost is low (only a single calculation per time-step) and it is better at energy conservation for Lennard-Jones(LJ) type potentials than higher-order methods[2].

It can be derived as follows. Assuming we're given initial conditions for positions and velocities of particles, then we wish to find the position of a particle at the next time step, $x(t+h)$ where $h \equiv \Delta t$ and t is the current time. We can take a Taylor expansion of $x(t+h)$ about t to get

$$\begin{aligned} x(t+h) &= x(t) + \dot{x}(t)(t+h-t) + \ddot{x}(t)\frac{(t+h-t)^2}{2} + O(h^3) \\ &= x(t) + h\dot{x}(t) + \frac{h^2}{2}\ddot{x}(t) + O(h^3) \end{aligned} \tag{3.1}$$

where we have a truncation error of order $O(h^3)$. We can do the same for $x(t-h)$

$$x(t-h) = x(t) - h\dot{x}(t) + \frac{h^2}{2}\ddot{x}(t) - O(h^3) \tag{3.2}$$

Now adding equations 3.1 and 3.2 together we get

$$x(t+h) + x(t-h) = 2x(t) + h^2\ddot{x}(t) + O(h^4) \tag{3.3}$$

which can be re-arranged to

$$x(t+h) = 2x(t) - x(t-h) + h^2\ddot{x}(t) + O(h^4) \tag{3.4}$$

So we can get the next position from the current position and the last position. $\ddot{x}(t)$ is known from the force calculations from the potentials of the system[2]. This will be discussed in the next section. Now the velocity is not required for this calculation but can be gotten from

$$\dot{x}(t) = [x(t+h) - x(t-h)]/2h + O(h^2) \tag{3.5}$$

Since we need the current and last positions to predict the next position this method is not self-starting. Also the current velocity must be calculated separately. There is a form of the Verlet algorithm known as the Velocity Verlet that solves both of those issues. We can get it by first rearranging equation 3.5 to solve for $x(t-h)$,

$$x(t-h) = x(t+h) - 2h\dot{x}(t) \tag{3.6}$$

now substituting this into equation 3.4 we get

$$x(t+h) = x(t) + h\dot{x}(t) + \frac{1}{2}h^2\ddot{x}(t) + O(h^4) \tag{3.7}$$

This is how the next position is calculated for the Velocity Verlet algorithm. However, this is now dependent on the current velocity, so we need to calculate the velocity at the next step when we calculate the position at the next step. To do so we'll take equation 3.5 to write

$$\dot{x}(t+h) = [x(t+2h) - x(t)]/2h \tag{3.8}$$

Now using equation 3.4 in the case for $x(t+2h)$ and substituting in we get

$$\dot{x}(t+h) = [x(t+h) - x(t)]/h + \frac{1}{2}h\ddot{x}(t+h) \tag{3.9}$$

Next, equation 3.7 is substituted in and we finally get

$$\dot{x}(t+h) = \dot{x}(t) + \frac{1}{2}h(\ddot{x}(t+h) + \ddot{x}(t)) + O(h^2) \tag{3.10}$$

So now we have a self-starting algorithm that calculates both the position and velocity and is mathematically equivalent to the original Verlet algorithm[18].

3.2 Interaction Potentials

As mentioned in the last section we can get the next position of a particle from equation 3.7. In this equation we are required to know $\ddot{x}(t)$. This is the force on the particle from the interactions in the system. These interactions are modeled by various potentials, and in this work the potentials are interactions between particles. One of the most common potentials used in MD simulations is the Lennard-Jones potential, given by

$$U = \begin{cases} 4\epsilon\left[\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6}\right] & r < r_c \\ 0 & r \geq r_c \end{cases} \tag{3.11}$$

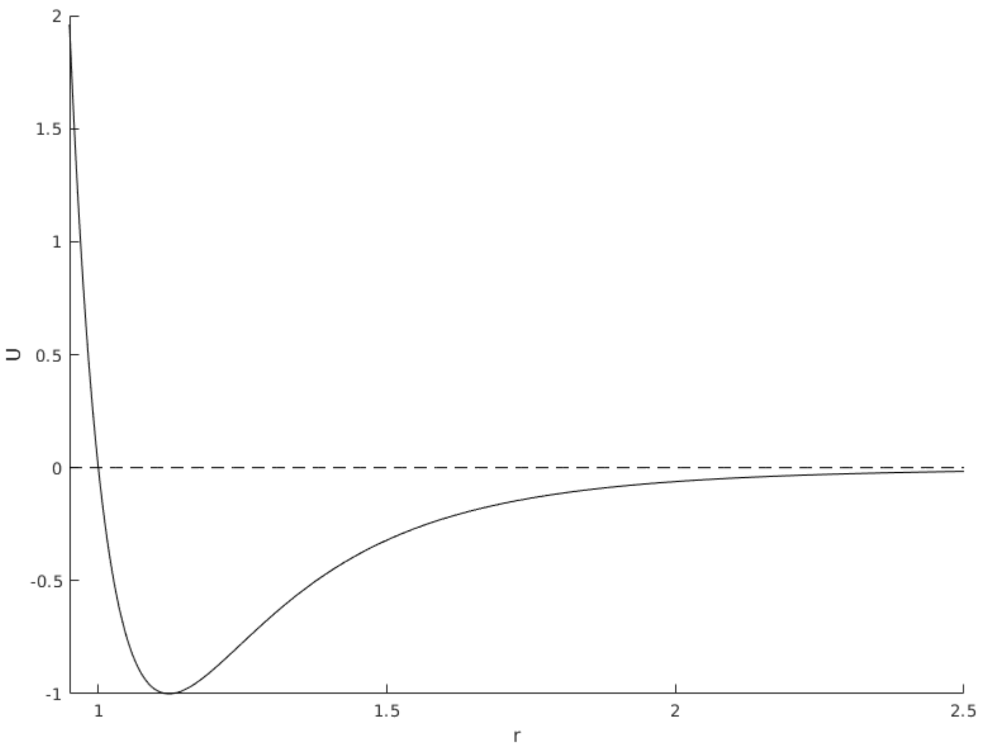


Figure 3.1: Graph of Lennard-Jones potential as a function of some arbitrary distance r from a particle vs arbitrary energy U.

This potential was initially determined empirically to describe the mechanics of liquid Argon[2], but has been found to work well for many liquid systems. σ refers to the diameter of the particle and ϵ determines the well depth. The first term, the twelfth power term, gives particles a hard-core nature. This comes about from the Pauli-Exclusion Principle[19] since the particles are distinguishable and cannot occupy the same states. The second term, the sixth power term, refers to the Van der Waals interactions between particles. As can be seen in Figure 3.1, this results in an attractive tail. r_c refers to the cutoff distance of the potential. As seen in Figure 3.1 it approaches zero fairly quickly as it is a short-range potential. A cutoff distance is introduced so particles outside this range do not interact, decreasing time needed to compute the interactions.

There are many potentials that can be used, like the Coulomb potential, ion-dipole interactions, bead-spring potentials, etc. Potentials can also be constructed based on empirical findings or from quantum calculations. The potentials chosen almost completely define the simulation and systems can be very sensitive to the nature of the interactions.

For a given potential, the force can be calculated by

$$F = -\nabla U \tag{3.12}$$

so for our example of the Lennard-Jones potential

$$F = \left(\frac{48\epsilon}{\sigma^2}\right)\left[\left(\frac{\sigma}{r}\right)^{14} - \frac{1}{2}\left(\frac{\sigma}{r}\right)^{8}\right]r \tag{3.13}$$

thus this divided by mass would be plugged into equation 3.7 for $\ddot{x}(t)$.

3.3 Boundary Conditions

Due to the finiteness of computers, MD simulations are limited in the number of particles, time to run simulations, and how large a system can be, due to computational cost and storage. It is impossible to simulate an infinite system of particles, or is it?

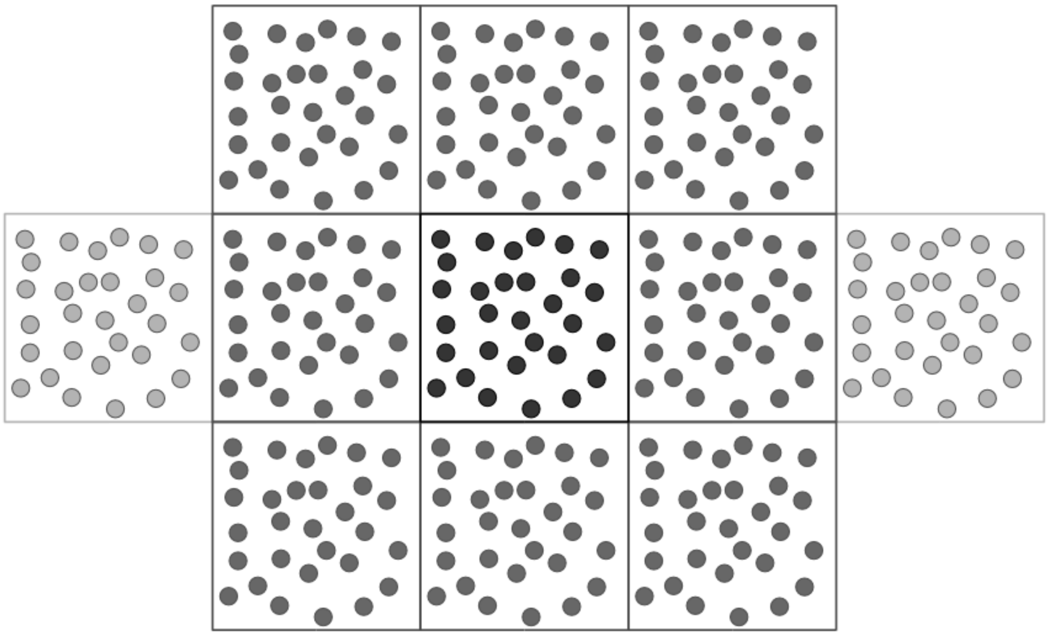


Figure 3.2: 2-D representation of periodic boundary conditions. Retrieved from Ref.[2]

Boundary conditions determine how the edges of a simulation box acts. They can be closed, like a solid wall that nothing is allowed to pass. It may transfer energy or heat to the system. Or the system can be periodic, meaning when a particle passes through the edge of the box it reemerges on the other side. It can either be thought of like a game of Pac-man, or visualized as an infinite series of identical boxes extending on all sides of the simulation box as pictured in Figure 3.2. With this condition we can get closer to simulating bulk systems and better take into account the effects of long-range interactions. However, dealing with long-range interactions is actually more tricky than simply applying periodic boundaries. For example, when dealing with the long-range Coulomb interaction we must apply a technique known as an Ewald Summation.

The Ewald Summation technique attempts to calculate the total interaction energy from the contributions of all the periodic systems. For N_m charged atoms the total interaction energy for charges is

$$U_{qq} = \frac{1}{2} \sum_{\vec{n}}{}' \sum_{i=1}^{N_m} \sum_{j=1}^{N_m} \frac{q_i q_j}{\vec{r_{ij}} + L\vec{n}} \tag{3.14}$$

where $\vec{n}$ are the integer vectors, and the primed sum that sums over them omits terms where $i = j$ when $|\vec{n}| = 0$ so self-interaction doesn't occur. It will interact with its images in the replica boxes however. L is the length of a side of the simulation box. The Ewald Summation changes this sum over the replicas into a sum over concentric spherical shells, given by

$$
\begin{aligned}
U_{qq} = \sum_{i<j\leq N_m} q_i q_j \Bigg[& \sum_{\vec{n}}{}' \frac{erfc(\alpha|\vec{r_{ij}} + L\vec{n}|)}{|\vec{r_{ij}} + L\vec{n}|} \\
& + \frac{1}{\pi L} \sum_{\vec{n}\neq 0} \frac{1}{n^2} exp\Big(- \frac{\pi^2 n^2}{\alpha^2 L^2} + \frac{2\pi i}{L} \vec{n} \cdot \vec{r_{ij}} \Big) \Bigg] \\
& + \frac{1}{2} \Bigg[\sum_{\vec{n}\neq 0} \Bigg(\frac{erfc(\alpha L n)}{Ln} + \frac{1}{\pi L n^2} exp\Big(- \frac{\pi^2 n^2}{\alpha^2 L^2} \Big) \Bigg) - \frac{2\alpha}{\sqrt{\pi}} \Bigg] \sum_{j=1}^{N_m} q_j^2 \\
& + \frac{2\pi}{3L^3} \Bigg| \sum_{j=1}^{N_m} q_j \vec{r_j} \Bigg|^2
\end{aligned} \tag{3.15}
$$

where the complementary error function is

$$
erfc(x) = \frac{2}{\sqrt{\pi}} \int_x^{\infty} e^{-t^2} dt \tag{3.16}
$$

Here, α is a parameter that is determined to maximize the accuracy of the sum. α can be chosen such that the terms of order $exp(-\alpha^2 L^2)$ are negligable so the real space terms become short-range allowing a cutoff distance of $r_c < L/2$ to be used along with period boundary conditions. The Ewald Summation can be used for dipoles as well. For more information on its formulation, *The Art of Molecular Dynamics* by D.C. Rapaport is an excellent resource[2].

3.4 Thermostat

Another issue stemming from limitations in computational storage is that of energy conservation in MD simulations. Computers cannot hold numbers of infinite precision so at some point there must be a cutoff or rounding. If done a couple of times it wouldn't be much of an issue, but when dealing with hundreds or thousands of particles for millions or more time steps the errors can accumulate greatly. This can result in runaway temperature or energy. To combat this issue we introduce a thermostat which regulates energy, temperature, pressure, volume, or any other quantities that need to be held at a near constant value.

An example of a very basic thermostat is the velocity re-scaling technique. This technique works by keeping track of the temperature by averaging the velocities of the particles. When the temperature begins to diverge from whatever preset value was chosen, it randomly picks particles and re-scales their velocities in order to correct the temperature. While simple, this technique is not necessarily the most efficient and does not allow for temperature fluctuations like that of a canonical ensemble.

Another thermostat commonly used is the Nose-Hoover thermostat. This thermostat treats the system as though it has a heat bath. With this heat bath an extra degree of freedom is introduced by means of an artificial variable associated with a mass and velocity. The mass determines the coupling of the heat bath with the system to affect the fluctuations in temperature. The main purpose of this thermostat is to treat the system as a canonical ensemble, as by creation MD simulations are microcanonical[20].

Another technique is to use Langevin dynamics. A simple explanation for this technique is to occasionally apply random forces to particles. This simulates stochastic interactions with particles, a kind of Brownian dynamics. In this work this technique was used in tandem with the Nose-Hoover thermostat.

3.5 LAMMPS

LAMMPS stands for Large-scale Atomic/Molecular Massively Parallel Simulator. It is a classical molecular dynamics simulation software developed by Sandia National Laboratories. Development began in the mid 1990s, originally written in Fortran. It's modern incarnation is a C++ rewrite that began in 2004 as a free and open-source project[21]. It is easy to use and has been utilized successfully in many scientific publications[22].

LAMMPS uses the velocity-verlet algorithm given in equations 3.7 and 3.10. It has many thermostats available, from the Nose-Hoover thermostat or Langevin, etc.[23] It also contains many potentials for use in solid-state physics, soft-matter, polymer physics, etc.[24] A basic example script is given in Appendix B that uses a Lennard-Jones potential in a simulation box with periodic boundary conditions and uses the Nose-Hoover thermostat.

When writing a LAMMPS script a choice of units must be given. Molecular Dynamics simulations generally have some kind of unitless values for use in calculations. Since simulations are done on the atomic/mesoscopic scales unit values can be very small and very large(e.g. 10^{-30} to 10^{30}). As mentioned ealier, computers have finite precision and these kinds of values are challenging to store without rounding or cutoffs. In order to make numbers more easily handled, a conversion of units can be done to keep numbers around unity. In this work, the LAMMPS lj units were utilized.

LJ units are designed to set the fundamental quantities to one. Four quantities are set to be normalized, and these are used to define the values for the remaining quantities. The fundamental quantities are mass m, σ, ϵ, and the Boltzmann constant k_B. It works best like this, first pick a common value for these units and set them to one. For example, if many of the particles in a simulation are 4.5Å in diameter then $\sigma = 4.5$Å. Now when implementing into the simulation we convert values using this fundamental unit into a reduced unit. So all particles have $\sigma^* = 4.5\text{Å}/\sigma = 1$. If another particle with diameter 3Å was introduced for example, then it's diameter in reduced units is $\sigma^* = 3\text{Å}/\sigma = 0.6666...$ The full conversion list is on the LAMMPS documentation website at `https://lammps.sandia.gov/doc/units.html`.

LAMMPS is a parallel MD simulator, this means it can utilize multiple CPU cores at once to do its calculations. This allows it to be much faster than conventional single core MD simulations. LAMMPS is able to calculate the forces on many particles simultaneously, reducing computational time significantly. A spatial decomposition algorithm is used that divides the simulation box into multiple domains, each for use by a different processor[12].

LAMMPS is an excellent classical MD simulation program for its vast amount of features and ease of use. It continues to be in development, and being open-source allows users to implement their own modifications. If a feature is not currently built into the official release, user projects and libraries can be created and added to LAMMPS with ease. This makes it a great choice for use as an MD simulation software.

Chapter 4

Methods

4.1 Stockmayer Fluid

The Stockmayer Fluid Model is a system consisting of Lennard-Jones spheres embedded with point dipole moments and/or point charges. This model was originally conceived by W.H. Stockmayer [13] and has been applied to computational simulations looking at dielectric properties of polar liquids[25, 26].

The interactions in a Stockmayer Fluid consists of a Lennard-Jones potential, dipole-dipole interactions, ion-dipole interactions, and ion-ion interactions. For this work the Lennard-Jones potential, given by equation 3.11, has a cutoff distance of $r_c = 2^{1/6}\sigma$. This substitution is called the WCA Potential, shown below in Figure 4.1. We only wish for the hard-core nature of the spheres to be used, since the attractive tail of the LJ potential comes from the Van der Waals forces which ultimately arise from dipole-dipole interactions which we will take into account explicitly.

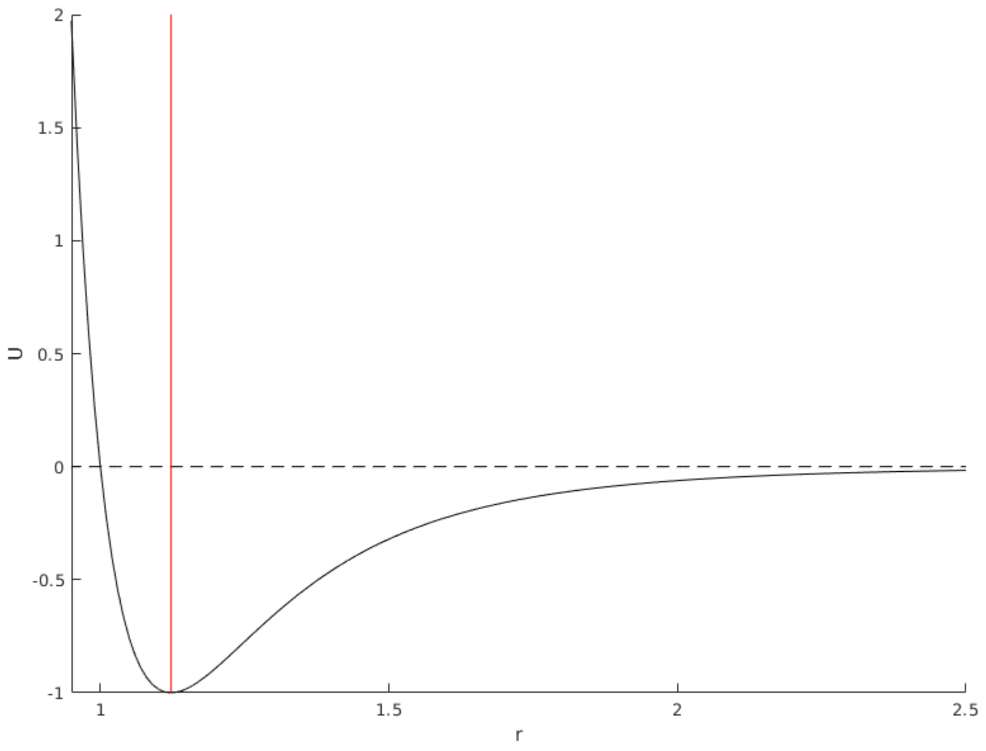


Figure 4.1: Graph of Lennard-Jones potential but the red vertical line indicates the cutoff distance for the WCA potential.

The dipole-dipole interaction potential is given by

$$U_{\mu\mu} = \frac{1}{r^3}(\vec{\mu_i} \cdot \vec{\mu_j}) - \frac{3}{r^5}(\vec{\mu_i} \cdot \vec{r})(\vec{\mu_j} \cdot \vec{r}) \tag{4.1}$$

the ion-dipole interaction is given by

$$U_{q\mu} = \frac{q}{r^3}(\vec{\mu} \cdot \vec{r}) \tag{4.2}$$

and finally the ion-ion interaction is simply the Coulomb potential

$$U_{qq} = \frac{q_i q_j}{r} \tag{4.3}$$

and these are all provided by the LAMMPS documentation[27].

Now while not done in the original inception of the Stockmayer Fluid Model, for this work we included polymerization of solvent particles. To model this we used the FENE potential given by

$$U_{FENE} = -0.5KR_0^2 \ln\left[1 - \left(\frac{r}{R_0}\right)^2\right] \tag{4.4}$$

This potential is essentially a bead-spring system, where two spheres are connected by a simple 1-D spring. K refers to the spring constant and R_0 is the cutoff radius of the spring. A value for K that's considered realistic is $K = 30\epsilon_{LJ}/\sigma^2$, and $R_0 = 1.5\sigma$[28].

4.2 Simulation Goals

The ultimate goal of this work has been to determine the efficacy of the Stockmayer Fluid Model for predicting solvation energies and comparing with the Born Solvation energy. As was shown in Chapter 2, the Born Solvation energy is a simple approximation that relies on treating the surrounding solvent molecules as a homogeneous dielectric material with some dielectric constant ϵ_r. Later the concept of dielectric saturation was introduced showing that in the near vicinity of the ions there is a lowering of the dielectric constant, distinct from that of the bulk value of the solvent. Next, the solvation shells surrounding the ion was also discussed, indicating that solvation energy relies nearly solely on the local structure of the dipolar molecules surrounding the ion. Thus because Born's equation doesn't take these effects into account it's bound to give quantitatively incorrect results. However, since the major underlying concepts are similar it may be qualitatively okay. This has been known for some time and there has been work on constructing a better theory.

One such construction is a dipolar self-consistent mean field theory (DSCFT) developed by Nakamura et al.[3] The specifics of the theory are beyond the scope of this work, but of notable importance is that when taking into account the local effects near the ion there is a raising of the solvation energy as compared to the Born Solvation energy, as shown in Figure 4.2.

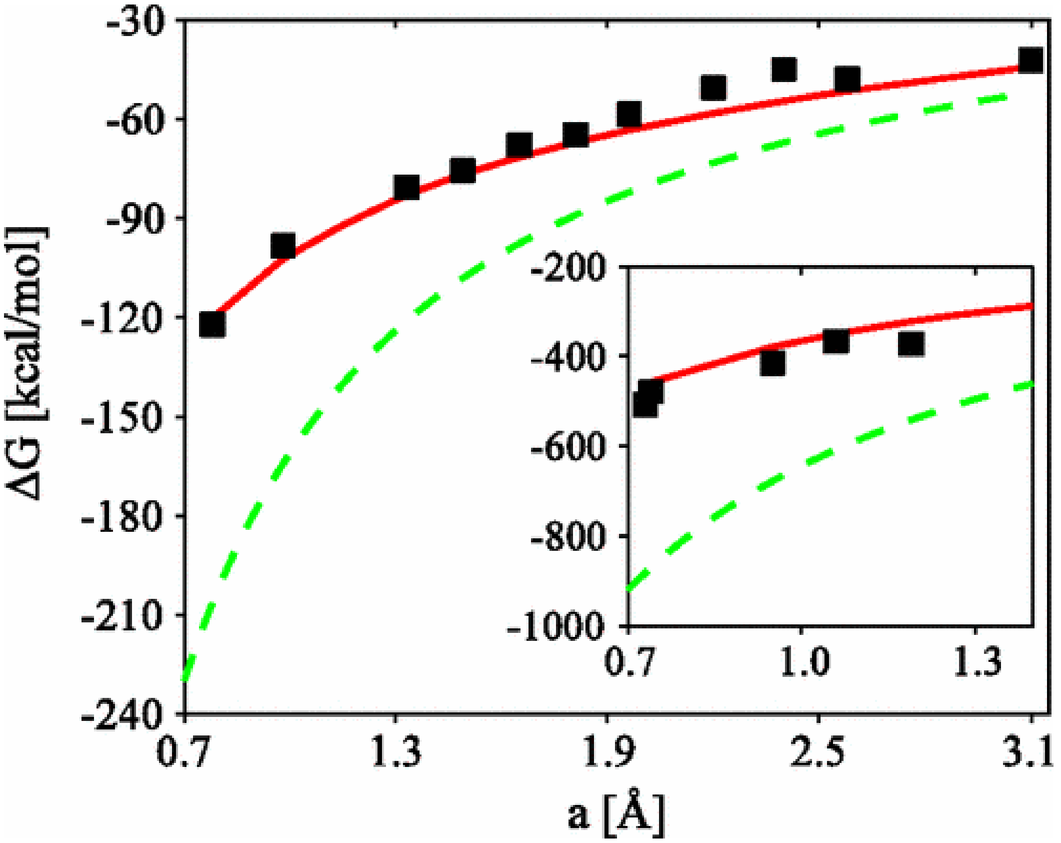


Figure 4.2: Graph comparing the Born Equation, DSCFT, and experimental values of solvation energy for ions of varying radius. The black dots are the experimental values, the solid red line is the solvation energy calculated by DSCFT, and the green dashed line is the Born Solvation energy. The outer graph plots monovalent ions while the inset plots divalent ions. Retrieved from Ref.[3]

So when simulated using a Stockmayer Fluid Model would the Born Solvation energy be seen or will the saturation effects be captured? What happens when the solvent is polymerized? And can experimental results be replicated with such a simple model? These are some of the questions that were asked as motivation for this work.

4.3 Method

This work was originally based on simulations constructed by Lijun Liu, a former member of Dr. Nakamura's research group[29]. Lijun created his own Molecular Dynamics program whereas for this work LAMMPS was used. For this work three separate sets of simulations were developed. The first was calculating the solvation energy of ions in a non-polymerized dipolar solvent with varying dipole moments. The second being the same simulation but with a polymerized solvent. Thirdly, the solvation energy for particular solvents and ions were calculated and compared to experimental results and DSCFT calculations.

While these simulations tested different systems, the underlying method was the same. A simulation box consisting solely of solvent dipoles was constructed and the total electrostatic energy was recorded over a period corresponding to 100ns. Next the same system but with the introduction of a single ion was ran, and again the energy was recorded. To calculate the solvation energy the average energy for the ion-containg system was subtracted from the average energy of the ion-free system. This is how the solvation energy was calculated for all systems.

To populate the simulation box with particles a Matlab script was written that would give particles random positions and random dipole orientations, as well as imbuing them with their other properties (mass, density, etc.). The script used for the first two systems is provided in Appendix C, and for the third system the script in Appendix D was used.

Minimization was done by giving all particles a soft potential and running for 1-2ns. A soft potential is similar to a Lennard-Jones potential in that the particles have a repulsive core, but there is a long gradual slope making the particle"soft" instead of hard-cored. This is done since the particles are given random initial conditions and could be placed inside of other particles. If the simulation was ran given all normal interactions the forces could be immensely large and in LAMMPS this leads to particles being lost. Thus the soft potential lets the particles slide off each other gradually and move some distance away before the real interactions of the system begin. A more detailed explanation of the soft potential is given by the LAMMPS documentation[30].

4.3.1 System: Non-polymerized Solvent

This system consisted of 512 solvent dipoles in a box of size 45Å × 45Å × 45Å. The diameter of the solvent was 4.5Å and for the ion 3Å. Both particles were given the same mass of 18 amu and the same WCA well depth of $\epsilon_{\mathrm{LJ}} = 5.27 \times 10^{-21}$J. The system temperature was set to 400K using the Nose-Hoover thermostat. For the time step, $\Delta t = 2\text{fs}$.

Ten systems in total were simulated, with solvent dipole moments from 0.5D to 5D in increments of 0.5D. The ions were simply given a point charge and a dipole moment of zero. The charge given to the ions were dependent on the type of ion tested. Monovalent, divalent, and trivalent ions were simulated, with charges of 1e, 2e, and 3e respectively. A set of example scripts are given in Appendix E.

4.3.2 System: Polymerized Solvent

This system is very similar to the previous, but now the solvent molecules are polymerized. The polymer chains have a chainlength of 16 monomers, giving a total of 32 polymer chains in the system. There is a caveat to polymerizing in LAMMPS however. Dipole rotation is independent of monomer rotation. This means that the dipoles moment of a particle can rotate freely and its rotation has no effect on the physical rotation of the particle itself. Another way to visualize would be to say that if the dipole were in a fixed direction, the monomer could still have a torque and rotate, and vice versa. This is important when discussing the polymerized solvent because the dipole moments can freely rotate along the polymer backbone. This is not realistic, but as of the moment of writing dipole-monomer rotation coupling is not implemented in LAMMPS. So this drawback must be taken into account when the results are discussed. A set of example scripts are given in Appendix F.

4.3.3 System: Experimental Replication

This set of simulations is currently a work in progress. Using the coarse-grained Stockmayer Fluid Model an attempt at recreating experimental data is being made. The following monovalent ions are being tested: Li^+, Na^+, Ag^+, K^+, Rb^+, Cs^+, Cl^-, Br^-, I^-, $ClO4^-$, $Me4N^+$, and $Et4N^+$. The following divalent ions are being tested as well: $Cu2^+$, $Zn2^+$, $Mg2^+$, $Mn2^+$, $Cd2^+$, $Ca2^+$, $Pb2^+$, $Sr2^+$, and $Ba2^+$. The radii of the ions and masses used in this study were consistent with those from Refs.[31] and [3] and are listed in Table 4.1. Currently the ions are placed in a coarse-grained methanol solvent. The dipole moment of the methanol is 1.69D[32] and two different coarse-grained sizes were used.

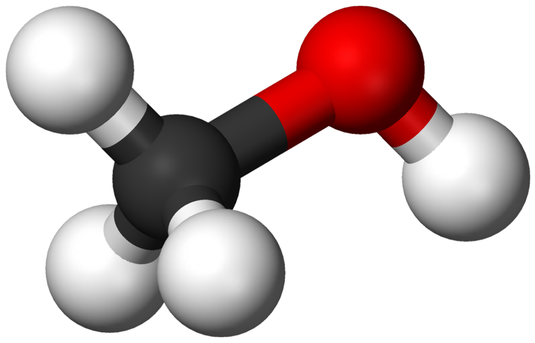


Figure 4.3: A computer model of a methanol molecule.Retrieved from Ref.[4]

The first size was determined by utilizing the program Gaussian. With Gaussian,

Species	Radius[Å]	Mass[a.m.u.]
Li^{+}	0.78	7
Na^{+}	0.98	23
Ag^{+}	1.15	108
K^{+}	1.33	39
Rb^{+}	1.49	85.5
Cs^{+}	1.65	133
Cl^{-}	1.81	71
Br^{-}	1.96	80
I^{-}	2.2	114.8
$ClO4^{-}$	2.4	99.5
$Me4N^{+}$	2.58	109.6
$Et4N^{+}$	3.1	147.26
$Cu2^{+}$	0.73	63.5
$Zn2^{+}$	0.74	65.4
$Mg2^{+}$	0.78	24.3
$Mn2^{+}$	0.91	54.9
$Cd2^{+}$	0.95	112.4
$Ca2^{+}$	1.06	40.1
$Pb2^{+}$	1.19	207.2
$Sr2^{+}$	1.27	87.6
$Ba2^{+}$	1.43	137.3

Table 4.1
Properties of ions for use in simulation.

the methanol molecule was constructed and then optimized using the B3LYP DFT method. To determine diameter, the distance between the two farthest atoms was found. This was the distance between the two farthest hydrogen atoms, which was found to be 2.9Å. Rigor was not of importance here because the purpose was to get a rough estimate of size, not to perfectly encapsulate the features of methanol. This method resulted in a fairly large sphere size, since as can be seen in Figure 4.3 if the molecule were encased in a sphere there would be quite a bit of empty space. To account for this a second size was selected that was slightly smaller, 2.6Å was decided. A set of example scripts are given in Appendix G.

4.4 Equipment Used

The equipment used to run these simulations mainly consisted of the Sky Bridge supercomputer cluster from Sandia National Laboratories. Sky Bridge nodes contain 64GB of RAM and two 2.6 GHz Intel Sandy Bridge 8-Core processors supplied by Cray[33]. Also used was the Superior Supercomputer from Michigan Technological University. The specifications for Superior are available on the MTU website, `https://hpc.mtu.edu/boilerplate/`. For testing purposes a workstation was used consisting of a 2.2 GHz Intel Xeon E5-2699 44-Core CPU with 32GB of RAM graciously supplied by Dr. Issei Nakamura.

The version of LAMMPS used was the 22 August stable release. Matlab Version 9.4.0.813654 (R2018a) was used to code the initial condition scripts. Python Version 2.7.5 was used for programs to conduct analysis.

Chapter 5

Results and Analysis

5.1 Radial Distribution Function

Before beginning the analysis of the results an analysis technique used in this work needs to be introduced. The radial distribution function, $g(r)$, is a function that measures the probability of finding particles radially outward from some chosen particle, or averaged over many particles. This can give a picture of the distribution of particles in the system as well as changes in density. The local density of a system can be calculated using the radial distribution function as

$$\rho(r) = \rho_{\text{bulk}} g(r) \tag{5.1}$$

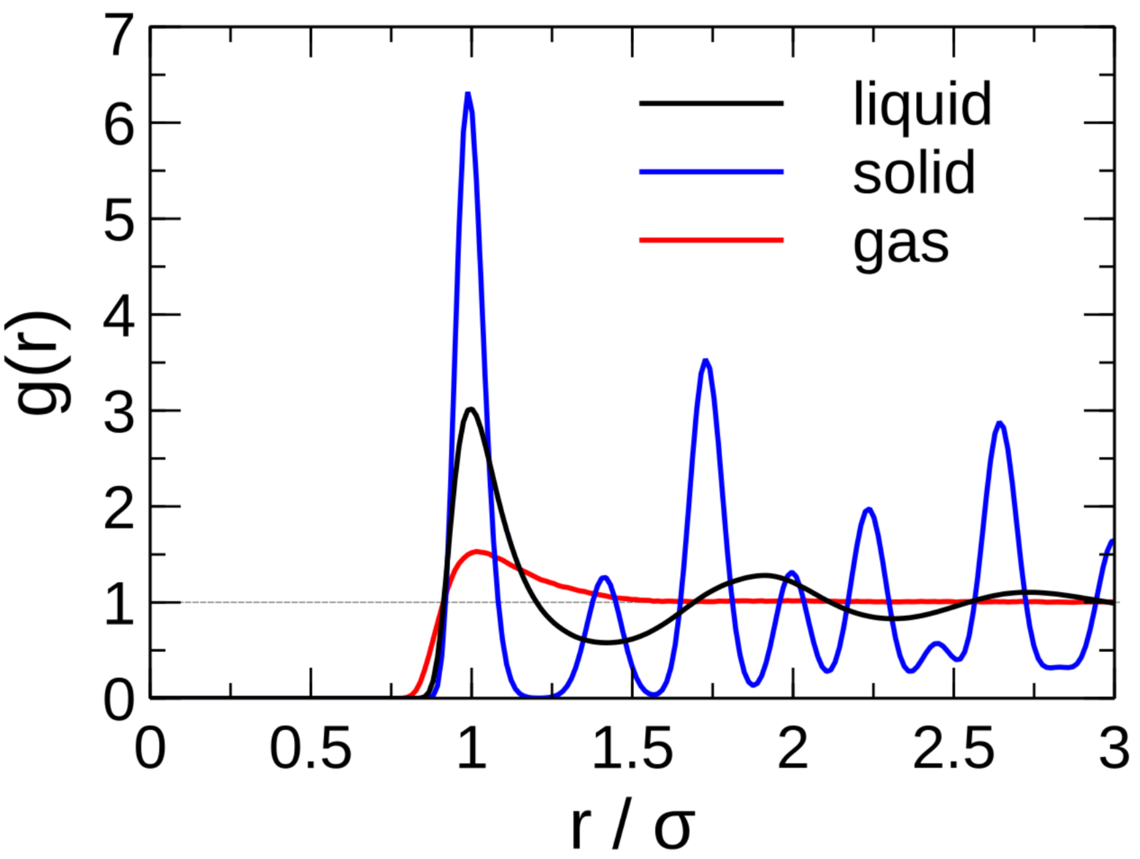


Figure 5.1: Example graph of the radial distribution function g(r) vs some normalized distance. This shows examples of g(r) for solids, liquids, and gasses.Retrieved from Ref.[5]

To calculate the number of particles at some distance r' then

$$n(r') = 4\pi\rho \int_0^{r'} g(r) r^2 dr \tag{5.2}$$

[34]

Figure 5.1 shows an example of a $g(r)$ graph for solids, liquids, and gasses. A solid usually consists of a crystal or regular lattice so it's expected that the $g(r)$ graph shows a periodic pattern. For a liquid we have the various shells surrounding the molecule so we witness several bumps before reaching $g(r) = 1$ for the bulk state. And for a gas the bulk state is reached quickly as generally gas particles don't interact.

5.2 System: Solvation Energy of Polymerized and Non-polymerized Solvent

The first two systems mentioned in Chapter 4 were done together and compared. The purpose was to see how well the Born Solvation energy could describe the Stockmayer Fluid. Also, since Born Solvation energy doesn't discriminate between a polymerized or non-polymerized system, would both give the same energy or would there be a discrepancy? The Born Solvation energy was calculated by equation 2.5 using ϵ_r calculated from Onsager's Equation, given by equation 2.72, using the properties of the dipolar solvent for each simulation. Figure 5.2 shows the results for a monovalent ion.

As can be seen from the graph, at low dipole moments the solvation energies for the polymerized and non-polymerized solvents are nearly the same. However, the Born Solvation energy is much lower. This is to be expected due to dielectric saturation effect which the Born Solvation energy does not take into account. Since dielectric saturation causes a decrease in the dielectric constant near the ion an increase in the

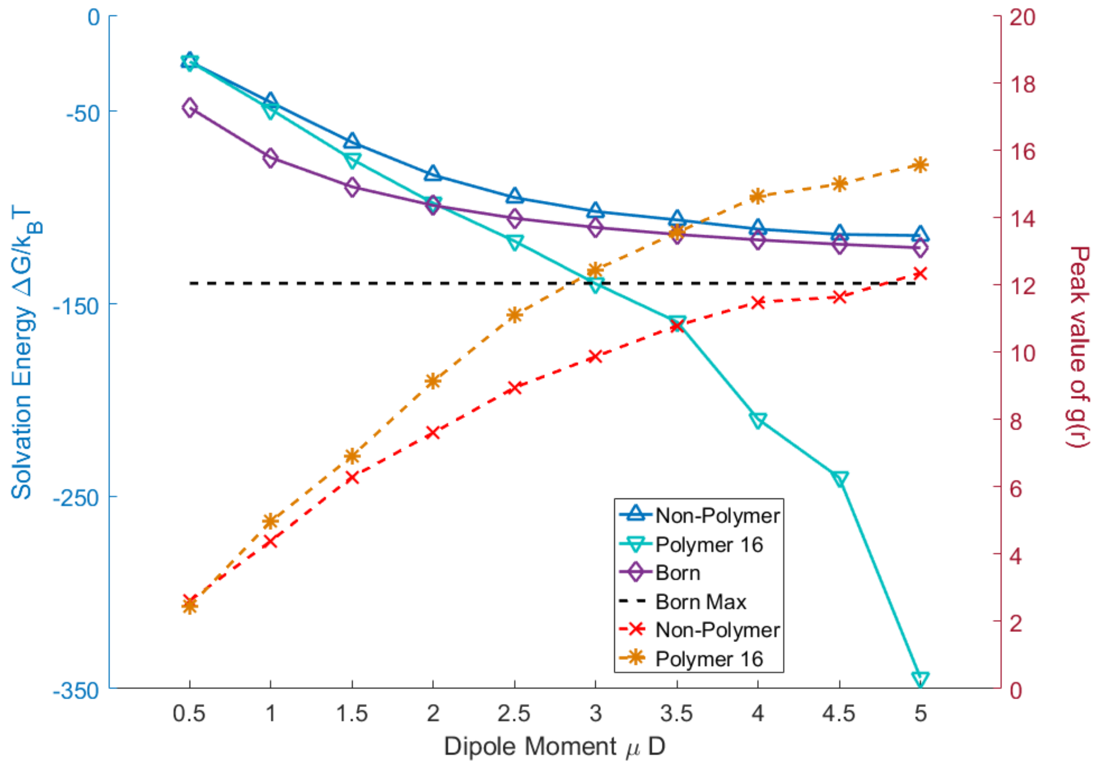


Figure 5.2: Graph of the solvation energy of a monovalent ion in a dipolar solvent at 400K as function of dipole moment of solvent molecules. The blueish downward sloping lines refer to the left-side y-axis and the reddish upward sloping lines refer to the right-side y-axis. Non-polymer refers to the case of the non-polymerized solvent, Polymer 16 refers to the polymerized solvent (chainlength of 16 monomers), Born refers to the predicted Born Solvation energy, and Born Max refers to the max value the Born Solvation energy can have.

solvation energy is expected. Of note here is that in the non-polymerized case, the solvation energy and the Born Solvation energy follow a similar trend. Toward higher dipole moments the simulated solvation energy approaches the predicted Born energy. As is expected, the trends are similar in that the energies decrease with increasing dipole moment(e.g. higher dielectric constant). The limit such that $\epsilon_r \to \infty$ is plotted as the black horizontal line, and its importance will be discussed later in this chapter.

Of particular interest is that the trend for the polymeric solvent is quite unlike that for the predicted Born energy or non-polymeric solvent, exhibiting linear behavior instead of approaching some limit. As of current all that can be offered for explanation is just speculation, as more work would need to be done to determine precisely what is the reason. The current thought is that since the monomers are chained together, the dipoles are forced near each other. This results in more dipoles being aligned than in the case of the non-polymerized solvent. Next, the severe drop in the large dipole moment regime is very puzzling. It is currently believed to be the result of either poor statistical convergence of the simulation, or is the result of highly correlated dipoles extending beyond the simulation box. Perhaps a larger box size would reduce this drop. As was mentioned, more work needs to be done in order to determine which of these outcomes is truly the case.

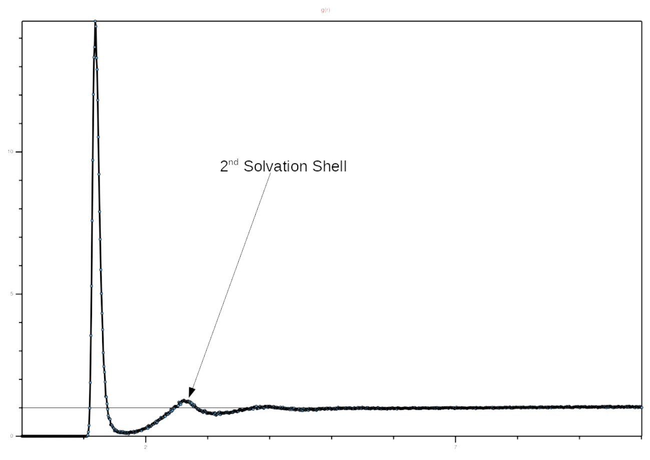
What cannot be overlooked is the upward trend of the peak value of the radial distribution function $g(r)$. In the lower dipole regimes ($\mu \leq 3.5$) this clearly corroborates what is occuring with the solvation energies. Increasing peak $g(r)$ means more solvent dipoles surrounding the ion in the first solvation shell, resulting in a higher dielectric constant thus decreasing the solvation energy. Also it can be seen that the peak values of $g(r)$ diverge for the polymeric and non-polymeric solvents which is what would be expected of diverging solvation energies. It also makes sense that the non-polymeric solvent peak value of $g(r)$ would be less than that of the polymeric solvent since the solvation energy is higher. Again what is odd is what is seen at the high dipole moment regime for the polymeric solvent. From the drastic drop in solvation energy we would expect a drastic increase in $g(r)$, but this is not seen. It was thought that perhaps the first solvation shell was saturated and this increase was the result of an increase of particles in the second solvation shell. However, as can be seen in Figure 5.3 there is no drastic increase, so this further reaffirms the belief that the decrease is due to some error or a long range orientational order of dipoles.

As mentioned in Chapter 4, divalent and trivalent ions were also simulated. Their results are provided in Figure 5.4.

There are obvious similarities between these graphs and that of the monovalent ion, but also some noticeable and some subtle differences. Most clear is that again the solvation energy for the non-polymerized solvent is above the Born Solvation energy due to dielectric saturation. Also, both seem to be approaching the same value and exhibit qualitative similarities. However, as expected, the values of the solvation energy are different, significantly lower in fact, in comparison to the monovalent ion. This is from the increase in ion charge and also its effect on the surrounding solvent dipoles.

With regards to the differences, it can be seen that in the low dipole moment regime the polymeric solvent and the non-polymeric solvent are now significantly different. This is very interesting and was quite unexpected. What makes it particularly fascinating is when looking at the peak values of $g(r)$ in this region. The values for

[$g(r)$ at 4.0D]



[$g(r)$ at 5.0D]

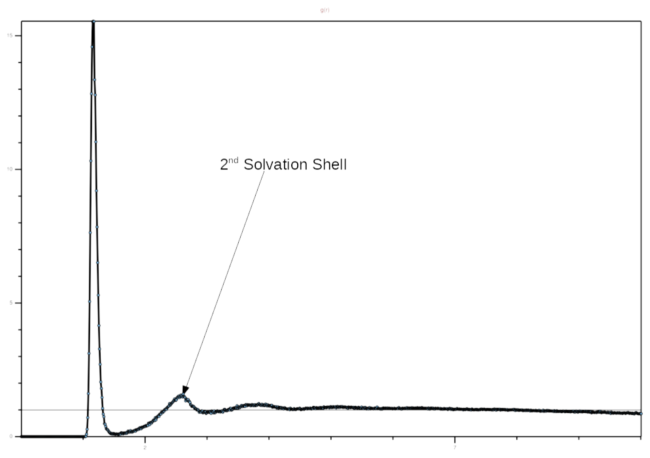
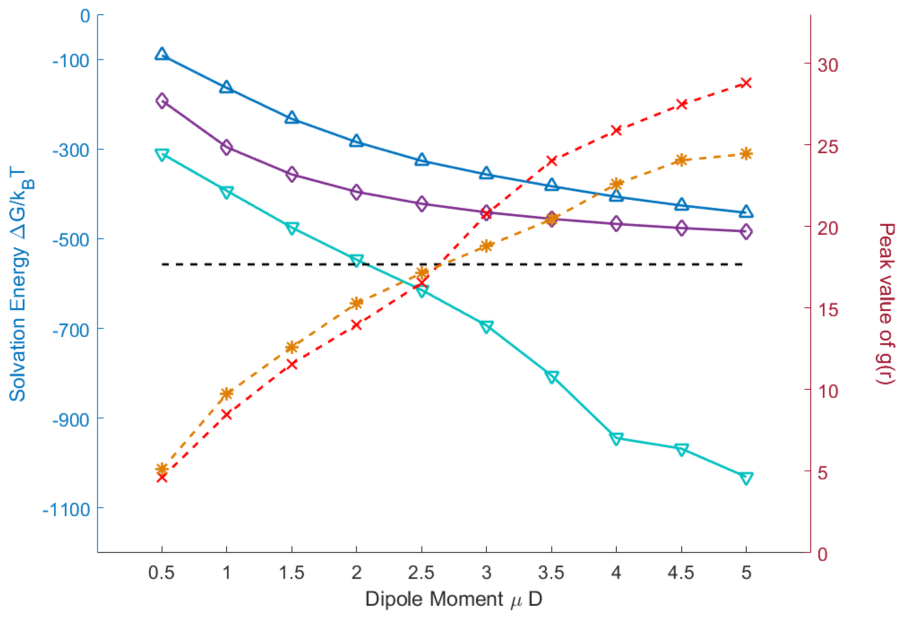


Figure 5.3: $g(r)$ graphs for polymerized solvent comparing 2nd solvation shell.

the polymeric and non-polymeric cases are very similar to each other indicating they have a similar number of particles in the first solvation shell. Because of this it would be expected to see a similar value in solvation energy, not a divergence. Though as mentioned earlier, this may be due to the chaining of monomers forcing dipoles near each other, thus increasing the correlation of dipole orientation. And again it is seen

[Solvation Energy of Divalent Ion]



[Solvation Energy of Trivalent Ion]

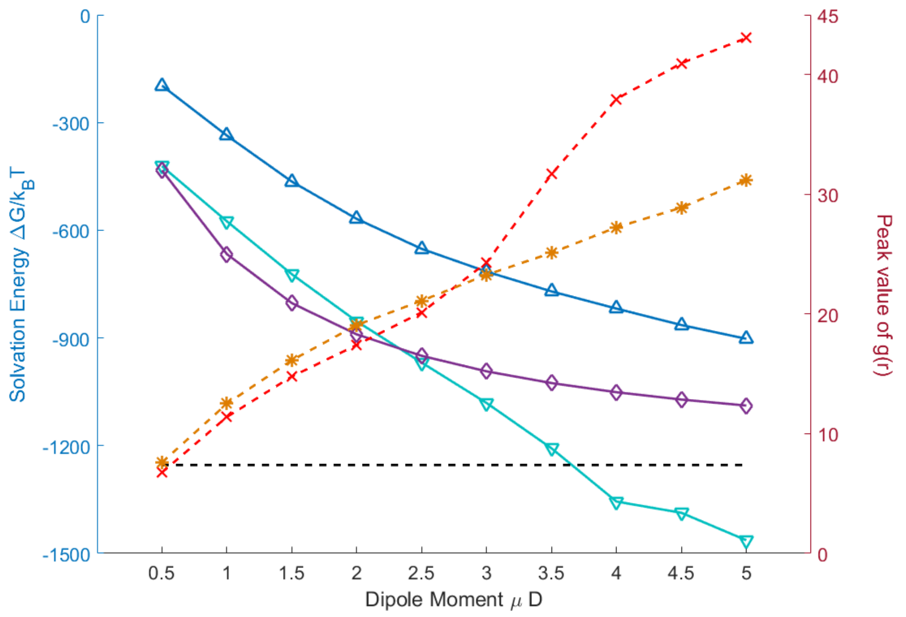


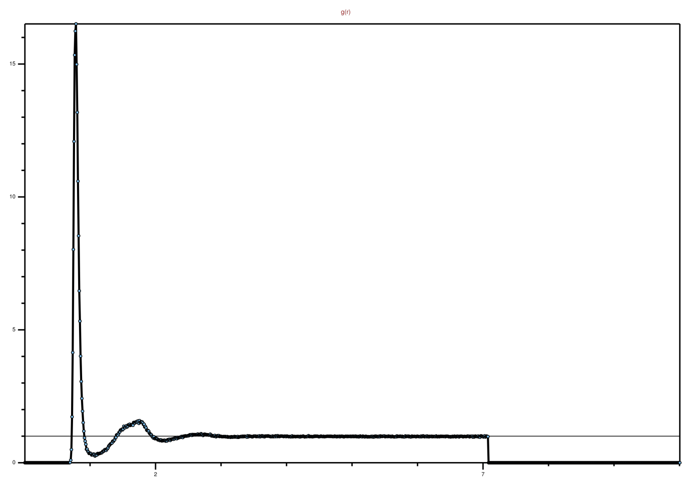
Figure 5.4: Solvation energy graphs for divalent and trivalent ions at 400K. Description of elements and legend given by Figure 5.2.

that the polymeric solvent exhibits a linearly negative slope, at odds with that of the non-polymeric solvent and predicted Born Solvation energy.

Now another interesting aspect of the divalent and trivalent graphs is the "swapping" of the peak values of $g(r)$ between 2.5D and 3D of the polymeric and non-polymeric solvents. In both cases there seems to be a dramatic increase in solvent particles in the first solvation shell for the non-polymeric solvent. With this dramatic jump it would be expected to see some noticeable effect on the solvation energy, however there is no such effect seen. The reason for this sudden increase is still currently unknown, more work needs to be done to determine its cause. What can be commented on is a proposed reason as to why the non-polymeric solvent would exhibit this behavior and is absent for the polymeric case. The current idea is that since the monomers in the polymeric solvent are chained up they are restricted in their movement, making it harder for other monomers to get near to the ion. The non-polymeric solvent does not have these structural restrictions allowing freer movement and allowing for more dipoles near the ion. As to why this is not seen in the monovalent ion case may be because the force is not as great. The force from the divalent ion is 4x greater so it could be argued the monovalent ion simply isn't strong enough to show this behavior.

What makes the polymeric case even more interesting is to focus on the max value for the Born Solvation energy mentioned earlier. This is when $\epsilon_r \to \infty$ indicating the lowest possible value the Born Solvation energy could have for a given ion charge

[$g(r)$ Non-Polymer]



[$g(r)$ Polymer]

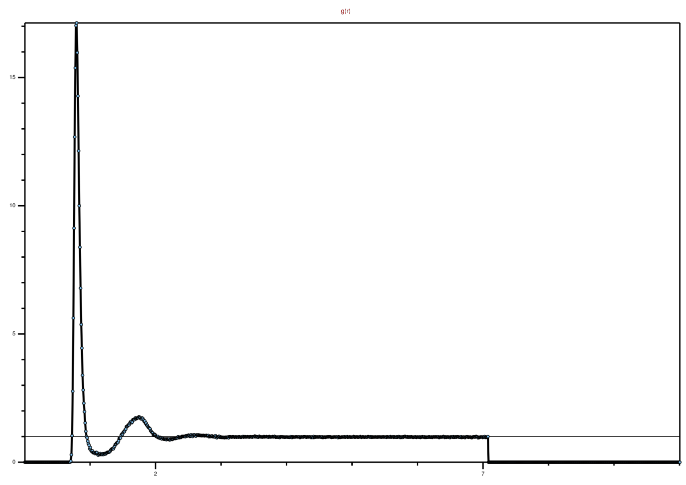


Figure 5.5: $g(r)$ graphs for polymerized and non-polymerized solvent at 2.5D for divalent ion case.

and size. One question that arose from this project was whether it was the Born Solvation energy that was incorrect, or whether Onsager may have been at fault. Could all of the important physics be captured by a correct understanding of the dielectric constant? Perhaps using a dielectric function would be more apt, so as to account for the dielectric saturation effect. The case for the polymeric solvent seems to show this is not the case, and that the Born Solvation energy may be missing something. Since in the higher dipole moment regimes of the monovalent, divalent, and trivalent cases it is seen that the polymeric solvent falls below this max value. Therefore there are effects not explained by changes in the dielectric constant. If looking solely at the Born Solvation energy, the only things that could account for this is an increased charge or a smaller ion size. Increased charge is out of the question since that is held constant due to the ion being the only charge. The ion size could be the culprit here. Perhaps the dipoles are closing in on the ion more-so in the polymeric case than the non-polymeric case, resulting in an effectively smaller cavity, smaller ion. Looking at the comparisons of $g(r)$ in Figure 5.5 however indicate no significant change in distance from the ion. Thus something not accounted for in the Born Solvation energy is occurring. One thought is that the compressibility of the solvent could play a part. A paper published by Nakamura shows that with increasing compressibility the solvation energy lowers, shown in Figure 5.6[6].

Born Solvation energy assumes an incompressible liquid, so taking compressibility into account could lower the energy, but as seen in the figure this change is small so there may be something else at play. As was mentioned in Chapter 4, there are caveats to the implementation of polymerization in LAMMPS and this behavior could simply be a result of the model. Perhaps if monomer rotation and dipole orientation were coupled something different would be seen. At the moment there are people at Sandia National Laboratories working to implement this feature, so when that is completed we can compare.

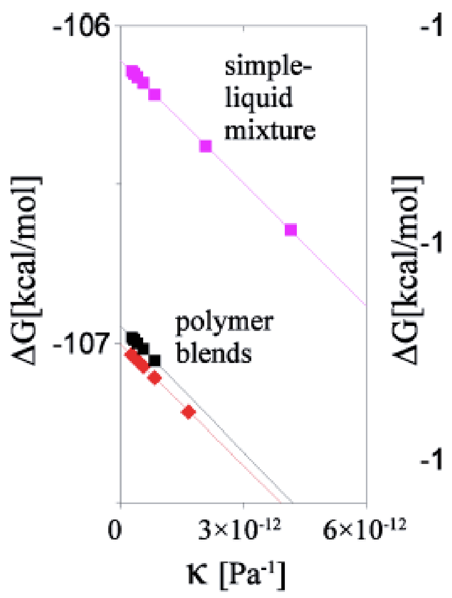


Figure 5.6: Graph showing that as compressibility of a solvent increases the solvation energy decreases.Retrieved from Ref.[6]

5.3 System: Experimental Replication

As mentioned in Chapter 4, a series of varyingly sized ions were solvated in methanol. Two different sizes for the coarse-grained methanol were used, 2.9Å and 2.6Å. The results were plotted in Figure 5.7 along with experimental results[35, 36, 37, 38] and DSCFT calculations for divalent ions.

As can be seen, both sizes closely match experimental data. There exists a couple of outliers but otherwise this model explains the data very well. This result was very surprising given it was a preliminary trial that matched the data so closely. Because of this we can make several statements about the veracity of the Stockmayer

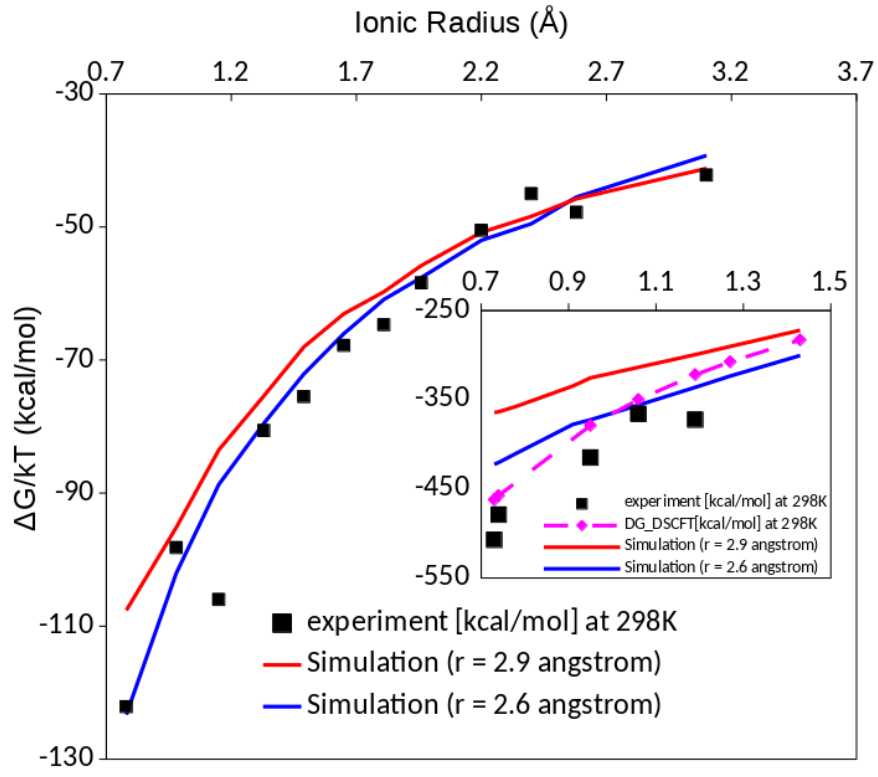


Figure 5.7: Graph of the solvation energy of ions of varying radii in methanol. The black squares refer to experimental results, the red line for the larger radii methanol, the blue line for the smaller radii methanol and the purple dashed line for DSCFT calculation. The outer graph shows monovalent ions and the inset shows divalent ions.

Fluid model with regards to solvation energy of ions. First, this shows that we can accurately explain solvation energy only in terms of ionic charge, size of molecules, and orientational polarization of permanent dipole moments. It appears that electronic polarization does not play much of a role, at least for monovalent ions. Looking at the divalent ions it may need to be considered, especially since the forces are 4x greater than that of a monovalent ion. Also this lends credence to the idea that the Stockmayer Fluid Model is a reasonable tool for modeling solvation of ions in dipolar solvents. This has not been shown before, and will be a good tool for the future, as the Stockmayer Fluid Model is a fairly simple model that is computationally faster to simulate than a full atomistic simulation. This also helps us reinforce and better understand the fundamental aspects of solvation, particularly what electronic properties matter most.

Now one simulated model is not enough to back up all those claims. Currently we are working to simulate many other solvents including ethanol, acetone, etc. As of writing we have received promising results, but those are to be talked about in another paper. The purpose of writing this here is to paint a picture of this method's application and that this model can actually tell us something about the world.

Chapter 6

Conclusions

The solvation of ions in solvents has been long studied and discussed. It could be argued that its behavior has been mostly understood since the mid-twentieth century. However, there are still many systems that have yet to be looked at and as has been shown in this work, much of what's understood may be qualitatively correct but quantitatively suffers. Scientists still invoke the Born Solvation energy to this day, yet we have shown that it has many shortcomings. Debye took great strides in improving our understanding of the nature of polar molecules, but he had to be corrected by Onsager, who has since been corrected by Kirkwood. Dielectric saturation has been understood since Debye and it is well known that the solvation energy is dependant almost entirely on the solvation shells surrounding an ion. Yet with the introduction of modern techniques like molecular dynamics we can better probe these theories and find in what situations they succeed and fail.

We showed that the Born Solvation energy doesn't take into account the dielectric saturation effect or invoke the idea of solvation shells. It assumes an incompressible homogeneous dielectric material, a very idealized situation which does not reflect the actual nature of the system. Through molecular dynamics using a simple Stockmayer Fluid Model we were able to demonstrate these effects and show that the Born Solvation energy fails to quantitatively describe the systems it claims to. We also showed that simply polymerizing solvent dipoles significantly changes solvation energy and may reveal the need to introduce new parameters to explain the concept of solvation energy. And finally, we showed from preliminary results that the Stockmayer Fluid Model may be enough to accurately simulate the effects of ion solvation in non-polymeric solvents.

This work shows that while we may already understand much on the nature of ion solvation, there is still more that needs to be worked out. What happens to the solvation energy when dipole rotation is coupled with monomer rotation in a polymerized solvent? Can the Stockmayer Fluid Model accurately predict solvation energies for all systems? To what extent does the compressibility of a solvent effect the solvation energy? Does the strength of the FENE potential have any influence on the solvation energy? These are just some of the questions that can still be asked, as there is so much we still don't know. This is a topic that has been studied well for over 100 years, and it may still be for 100 more.

References

[1] Gong, H.; Freed, K. F. *Physical review letters* **2009**, *102*(5), 057603.

[2] Rapaport, D. C.; Rapaport, D. C. R. *The art of molecular dynamics simulation;* Cambridge university press, 2004.

[3] Nakamura, I.; Shi, A.-C.; Wang, Z.-G. *Physical review letters* **2012**, *109*(25), 257802.

[4] Benjah-bmm27. *Methanol-3D-balls.png;* `https://commons.wikimedia.org/wiki/File:Methanol-3D-balls.png`.

[5] Rowley, C. *Simulated Radial Distribution Functions for Solid, Liquid, and Gaseous Argon;* `https://commons.wikimedia.org/wiki/File:Simulated_Radial_Distribution_Functions_for_Solid,_Liquid,_and_Gaseous_Argon.svg`.

[6] Nakamura, I. *The Journal of Physical Chemistry B* **2014**, *118*(21), 5787–5796.

[7] Von Born, M. *Zeitschr. Physik* **1920**, *1*, 45–48.

[8] Debye, P. *Polar Molecules, by P. Debye.*

[9] Onsager, L. *Journal of the American Chemical Society* **1936**, *58*(8), 1486–1493.

[10] Booth, F. *The Journal of Chemical Physics* **1951**, *19*(4), 391–394.

[11] Samoilov, O. Y. *Izd. Akad. Nauk SSSR, Moscow* **1957**, pages 76–102.

[12] Plimpton, S. *Journal of computational physics* **1995**, *117*(1), 1–19.

[13] Stockmayer, W. *The Journal of Chemical Physics* **1941**, *9*(12), 863–870.

[14] Rashin, A. A.; Honig, B. *The journal of physical chemistry* **1985**, *89*(26), 5588–5593.

[15] Raju, G. *Dielectrics in Electric Fields*, Power engineering; Taylor & Francis, 2003.

[16] Booth, F. *The Journal of Chemical Physics* **1955**, *23*(3), 453–457.

[17] Kuznetsov, V.; Usoltseva, N.; Zherdev, V. *Russian Journal of Inorganic Chemistry* **2014**, *59*(6), 637–642.

[18] Nikolic, B. *Verlet Method;* `http://www.physics.udel.edu/~bnikolic/teaching/phys660/numerical_ode/node5.html`, 2003.

[19] Jones, R. A. *Soft condensed matter*, Vol. 6; Oxford University Press, 2002.

[20] Rühle, V. *Am. J. Phys* **2007**.

[21] `https://lammps.sandia.gov/`.

[22] `https://lammps.sandia.gov/papers.html`.

[23] `https://lammps.sandia.gov/doc/Howto_thermostat.html`.

[24] `https://lammps.sandia.gov/doc/pair_style.html`.

[25] Stevens, M. J.; Grest, G. S. *Physical Review E* **1995**, *51*(6), 5976.

[26] Gao, G.; Zeng, X. C.; Wang, W. *The Journal of chemical physics* **1997**, *106*(8), 3311–3317.

[27] `https://lammps.sandia.gov/doc/pair_dipole.html`.

[28] `https://lammps.sandia.gov/doc/bond_fene.html`.

[29] Liu, L.; Nakamura, I. *The Journal of Physical Chemistry B* **2017**, *121*(14), 3142–3150.

[30] `https://lammps.sandia.gov/doc/pair_soft.html`.

[31] Nakamura, I. *The Journal of Physical Chemistry B* **2018**, *122*(22), 6064–6071.

[32] `https://en.wikipedia.org/wiki/Methanol`.

[33] `https://hpc.sandia.gov/platforms/index.html`.

[34] Chandler, D. *Introduction to Modern Statistical Mechanics, by David Chandler, pp. 288. Foreword by David Chandler. Oxford University Press, Sep 1987. ISBN-10: 0195042778. ISBN-13: 9780195042771* **1987**.

[35] Burgess, J. *Metal ions in solution;* Halsted Press, 1978.

[36] Case, B.; Parsons, R. *Transactions of the Faraday Society* **1967**, *63*, 1224–1239.

[37] Abraham, M. H.; Liszi, J. *Journal of the Chemical Society, Faraday Transactions 1: Physical Chemistry in Condensed Phases* **1978**, *74*, 1604–1614.

[38] Bontha, J.; Pintauro, P. *The Journal of Physical Chemistry* **1992**, *96*(19), 7778–7782.

Appendix A

Onsager Equation Algebra

When introducing the new polarization given by equation 2.71 to the Classius-Mossotti equation 2.29, we get

$$\frac{\epsilon_r - 1}{\epsilon_r + 2} = \frac{N}{3\epsilon_0}\left(\alpha_e + \frac{\mu^2}{3k_BT + \mu R_m}\right) \tag{A.1}$$

Now by using equation 2.27 and the Lorentz-Lorenz equation 2.30 this can be rewritten as

$$\frac{\epsilon_r - 1}{\epsilon_r + 2} - \frac{n^2 - 1}{n^2 + 2} = \frac{N}{3\epsilon_0}\left(\frac{\mu^2}{3k_BT + \mu R_m}\right) \tag{A.2}$$

Plugging in the Reaction field from equation 2.62

$$\frac{\epsilon_r - 1}{\epsilon_r + 2} - \frac{n^2 - 1}{n^2 + 2} = \frac{N}{3\epsilon_0}\left(\frac{\mu^2}{3k_BT + \frac{2N\mu^2}{9\epsilon_0}\frac{(\epsilon_r-1)(n^2+2)}{(n^2+2\epsilon_r)}}\right) \tag{A.3}$$

Just looking at the right side, it can be reorganized to become

$$\begin{aligned} &= \frac{N}{3\epsilon_0}\left(\frac{\mu^2}{\frac{27k_BT\epsilon_0(n^2+2\epsilon_r)+2N\mu^2(\epsilon_r-1)(n^2+2)}{9\epsilon_0(n^2+2\epsilon_r)}}\right) \\ &= \frac{3N\mu^2(n^2+2\epsilon_r)}{27k_BT\epsilon_0(n^2+2\epsilon_r)+2N\mu^2(\epsilon_r-1)(n^2+2)} \end{aligned} \tag{A.4}$$

Next, multiplying the denominator to the other side we get

$$\Big[27k_BT\epsilon_0(n^2+2\epsilon_r)+2N\mu^2(\epsilon_r-1)(n^2+2)\Big]\left[\frac{\epsilon_r - 1}{\epsilon_r + 2} - \frac{n^2 - 1}{n^2 + 2}\right] = 3N\mu^2(n^2+2\epsilon_r) \tag{A.5}$$

Dividing by the right side gives

$$\left[\frac{9k_BT\epsilon_0}{N\mu^2} + \frac{2(\epsilon_r - 1)(n^2+2)}{3(n^2+2\epsilon_r)}\right]\left[\frac{\epsilon_r - 1}{\epsilon_r + 2} - \frac{n^2-1}{n^2+2}\right] = 1 \tag{A.6}$$

Now we'll divide by the second bracketed term and subtract the second term in the first bracket to write

$$\frac{9k_BT\epsilon_0}{N\mu^2} = \frac{1}{\left[\frac{\epsilon_r-1}{\epsilon_r+2} - \frac{n^2-1}{n^2+2}\right]} - \frac{2(\epsilon_r - 1)(n^2+2)}{3(n^2+2\epsilon_r)} \tag{A.7}$$

For now we'll look at just the right hand side. The first term can be rewritten as

$$\begin{aligned}\frac{1}{\frac{(\epsilon-1)(n^2+2)-(n^2-1)(\epsilon_r+2)}{(\epsilon_r+2)(n^2+2)}} &= \frac{(\epsilon_r+2)(n^2+2)}{(\epsilon_r-1)(n^2+2)-(n^2-1)(\epsilon_r+2)} \\ &= \frac{(\epsilon_r+2)(n^2+2)}{\epsilon_r n^2 + 2\epsilon_r - n^2 - 2 - \epsilon_r n^2 - 2n^2 + \epsilon_r + 2} \\ &= \frac{(\epsilon_r+2)(n^2+2)}{3(\epsilon_r - n^2)}\end{aligned} \tag{A.8}$$

The whole right hand side is now

$$= \frac{(\epsilon_r+2)(n^2+2)}{3(\epsilon_r - n^2)} - \frac{2(\epsilon_r - 1)(n^2+2)}{3(n^2+2\epsilon_r)} \tag{A.9}$$

Pulling out like terms

$$\begin{aligned}
&= \frac{(n^2+2)}{3}\left(\frac{\epsilon_r+2}{\epsilon_r-n^2} - \frac{2(\epsilon_r-1)}{n^2+2\epsilon_r}\right) \\
&= \frac{n^2+2}{3}\left(\frac{(\epsilon_r+2)(n^2+2\epsilon_r)-2(\epsilon_r-1)(\epsilon_r-n^2}{(\epsilon_r-n^2)(n^2+2\epsilon_r)}\right) \\
&= \frac{n^2+2}{3(\epsilon_r-n^2)(n^2+2\epsilon_r)}\left[\epsilon_r n^2+2\epsilon_r^2+2n^2+4\epsilon_r-2\epsilon_2^2\epsilon_r n^2+2\epsilon_r-2n^2\right] \qquad \text{(A.10)} \\
&= \frac{n^2+2}{3(\epsilon_r-n^2)(n_2^2\epsilon_r)}(3\epsilon_r n^2+6\epsilon_r) \\
&= \frac{(n^2+2)^2}{(\epsilon_r-n^2)(n^2+2\epsilon_r)}
\end{aligned}$$

So now putting into the whole equation and taking the reciprocal of both sides gives

$$\frac{N\mu^2}{9k_B T\epsilon_0} = \frac{(\epsilon_r-n^2)(n^2+2\epsilon_r)}{(n^2+2)^2} \qquad \text{(A.11)}$$

and there we have the Onsager Equation.

Appendix B

Sample LAMMPS Script

A simple example script for LAMMPS treating particles as LJ spheres. First given is the LAMMPS script, then the atom data file.

B.1 script_simplelj.in

```
#####################################################
#                                                   #
#                                                   #
# Filename: script.in                       #
# Author: Cameron Shock, 2018                       #
#              #
# Execute the script through:                       #
# lmp_exe < script.in                       #
#                                                   #
#####################################################

# VARIABLES
variable fname index atoms_simplelj.txt
variable simname index simplelj

# Initialization
units     lj
boundary    p p p
atom_style    atomic
log       log.${simname}.txt
read_data   ${fname}

#Neighbor list info
neighbor    0.4 bin
neigh_modify  every 10 one 10000

######################################################
#Minimization

#Manual Minimization
pair_style    soft 1.225
pair_coeff    * * 60.0

velocity    all create 0.7859 1231
fix         1 all nvt temp 0.7859 0.7859 1.0
thermo_style  custom step temp etotal
thermo          10000
timestep    0.0028 #2fs
run       50000
unfix 1
```

```
#Built-in minimization
#min_style    cg
#minimize     0.0 1.0e-10 1000000 1000000
#####################################################
# Equilibration
pair_style    lj/cut 1.225
pair_coeff    * * 1.0 1.0



velocity    all create 0.7859 1231
fix         1 all nvt temp 0.7859 0.7859 1.0
thermo_style  custom step temp etotal ke
thermo          1000
timestep    0.0028 #2fs
run        1000000

unfix 1



print "All done"
```

B.2 atoms_simplelj.txt

```
# Model for

100 atoms
1 atom types
0.000000  10.000000 xlo xhi
```

```
0.000000  10.000000 ylo yhi
0.000000  10.000000 zlo zhi


Masses

1 18.000000


Atoms

1 1 0.596189  6.819719  0.424311
2 1 0.714455  5.216498  0.967300
3 1 8.181486  8.175471  7.224396
4 1 1.498654  6.596053  5.185949
5 1 9.729746  6.489915  8.003306
6 1 4.537977  4.323915  8.253138
7 1 0.834698  1.331710  1.733886
8 1 3.909378  8.313797  8.033644
9 1 0.604712  3.992578  5.268758
10  1 4.167995  6.568599  6.279734
11  1 2.919841  4.316512  0.154871
12  1 9.840637  1.671684  1.062163
13  1 3.724097  1.981184  4.896876
14  1 3.394934  9.516305  9.203320
15  1 0.526770  7.378581  2.691194
16  1 4.228356  5.478709  9.427370
17  1 4.177441  9.830525  3.014549
18  1 7.010988  6.663389  5.391265
19  1 6.981055  6.665279  1.781325
20  1 1.280144  9.990804  1.711211
21  1 0.326008  5.611998  8.818665
22  1 6.691753  1.904333  3.689165
23  1 4.607259  9.816380  1.564050
```

```
24  1 8.555228  6.447645  3.762722
25  1 1.909237  4.282530  4.820221
26  1 1.206116  5.895075  2.261877
27  1 3.846191  5.829864  2.518061
28  1 2.904407  6.170909  2.652809
29  1 8.243763  9.826634  7.302488
30  1 3.438770  5.840693  1.077690
31  1 9.063082  8.796537  8.177606
32  1 2.607280  5.943563  0.225126
33  1 4.252593  3.127189  1.614847
34  1 1.787662  4.228857  0.942293
35  1 5.985237  4.709243  6.959493
36  1 6.998878  6.385308  0.336038
37  1 0.688061  3.195997  5.308643
38  1 6.544457  4.076192  8.199812
39  1 7.183589  9.686493  5.313339
40  1 3.251457  1.056292  6.109587
41  1 7.788022  4.234529  0.908233
42  1 2.664715  1.536567  2.810053
43  1 4.400851  5.271427  4.574244
44  1 8.753716  5.180521  9.436226
45  1 6.377091  9.576939  2.407070
46  1 6.761223  2.890646  6.718082
47  1 6.951405  0.679928  2.547902
48  1 2.240400  6.678327  8.443922
49  1 3.444624  7.805197  6.753321
50  1 0.067153  6.021705  3.867712
51  1 9.159912  0.011511  4.624492
52  1 4.243490  4.609164  7.701597
53  1 3.224718  7.847393  4.713572
54  1 0.357627  1.758744  7.217580
55  1 4.734860  1.527212  3.411246
56  1 6.073892  1.917453  7.384268
57  1 2.428496  9.174243  2.690616
```

```
58   1 7.655000  1.886620  2.874982
59   1 0.911135  5.762094  6.833632
60   1 5.465931  4.257288  6.444428
61   1 6.476176  6.790168  6.357867
62   1 9.451741  2.089349  7.092817
63   1 2.362306  1.193962  6.073039
64   1 4.501377  4.587255  6.619448
65   1 7.702855  3.502180  6.620096
66   1 4.161586  8.419292  8.329168
67   1 2.564410  6.134607  5.822492
68   1 5.407393  8.699410  2.647790
69   1 3.180741  1.192145  9.398295
70   1 6.455519  4.794632  6.393170
71   1 5.447161  6.473115  5.438859
72   1 7.210466  5.224953  9.937046
73   1 2.186766  1.057983  1.096975
74   1 0.635914  4.045800  4.483729
75   1 3.658162  7.635046  6.278964
76   1 7.719804  9.328536  9.727409
77   1 1.920283  1.388742  6.962663
78   1 0.938200  5.254044  5.303442
79   1 8.611398  4.848533  3.934564
80   1 6.714311  7.412579  5.200525
81   1 3.477127  1.499973  5.860921
82   1 2.621453  0.444541  7.549333
83   1 2.427854  4.424023  6.877961
84   1 3.592282  7.363401  3.947075
85   1 6.834159  7.040474  4.423054
86   1 0.195776  3.308579  4.243095
87   1 2.702704  1.970538  8.217212
88   1 4.299214  8.877710  3.911830
89   1 7.691144  3.967915  8.085141
90   1 7.550771  3.773955  2.160189
91   1 7.904072  9.493039  3.275654
```

```
92  1 6.712644  4.386450  8.335006
93  1 7.688543  1.672535  8.619805
94  1 9.898722  5.144235  8.842810
95  1 5.880261  1.547523  1.998628
96  1 4.069548  7.487057  8.255838
97  1 7.899630  3.185242  5.340641
98  1 0.899507  1.117057  1.362925
99  1 6.786523  4.951770  1.897104
100 1 4.950058  1.476082  0.549741
```

Appendix C

Atom Generation Script Ver. 1

This Matlab script was used to create atom data for the first two systems mentioned in Chapter 4.

C.1 generateDipolesCharge2typerandbondssolv.m

```
clear
clc
atoms = 96; %# of particles
atomTypes = 2; %# of types
box = [0.0000,10.0000;0.0000,10.0000;0.0000,10.0000]; %↩
   box dimensions
```

```
masses = [18.00,18.00]; %the masses for each type of ↩
   particle
atomIdMatrix = (1:atoms)'; %matrix of id's for each ↩
   particle
atomTypeMatrix = ones(atoms,1);%create a 2 column matrix↩
    to store each type
atomChargeMatrix = ones(atoms,1);
atomMomentMatrix = zeros(atoms,3); %matrix for dipole ↩
   moments
atomDiameterMatrix = 2.3333.*ones(atoms,1); %matrix for ↩
   diameter of particles
atomDensityMatrix = ones(atoms,1); %matrix for density ↩
   of particles
moleculeMatrix =(1:atoms)';
offset = 0;

if(atomTypes == 2)
    offset = 0;
end

%for each dipole moment
for m1 = 0.5:0.5:5.0
    moment1 = m1*0.838326;
    %randomly generate direction for dipole moment of ↩
       cations
    for p = 1:(atoms)
       phi = 2*pi*rand();
       theta = pi*rand();
       x = moment1*sin(theta)*cos(phi);
       y = moment1*sin(theta)*sin(phi);
       z = moment1*cos(theta);
       atomMomentMatrix(p,:) = [x,y,z];
    end
```

```
%randomly generate particle positions
atomPos = ones(atoms,3);
for n = 1:atoms
    posX = box(1,2) * rand();
    posY = box(2,2) * rand();
    posZ = box(3,2) * rand();
    atomPos(n,:) = [posX, posY, posZ];
end

%BONDS
chainlength = 16;
if chainlength ~= 0
    bonds = (atoms-offset) / chainlength * (↩
       chainlength - 1);
    bondTypes = 2;
    bondIdMatrix = (1:bonds)';
    bondTypeMatrix = ones(bonds,1);
    bondTypeMatrix(bonds/2+1:bonds) = 2;
    bondAtomMatrix = ones(bonds,2);
    moleculeMatrix = ones(atoms,1);
    molecule = 1;
    missing = 0;
    for n = 1:atoms-offset
       if( mod(n,chainlength) ~= 0 )
           bondAtom1 = n;
           bondAtom2 = n+1;
           bondAtomMatrix(n-missing,:) = [bondAtom1 ↩
              bondAtom2];
           moleculeMatrix(n) = molecule;
       else
           moleculeMatrix(n) = molecule;
           molecule = molecule + 1;
           missing = missing + 1;
       end
    end

    bondMatrix = [bondIdMatrix bondTypeMatrix ↩
       bondAtomMatrix];
else
    bonds = 0;
    bondTypes = 0;
end

if chainlength ~= 0
    %random walk put together polymers
    for n = 1:atoms-offset
       if(n ~= 1 && mod(n-1,chainlength) ~= 0)
           safe = 0;
           while (safe == 0)
               r = 1;
               phi = 2*pi*rand();
               theta = pi*rand();
               x = r*sin(theta)*cos(phi);
               y = r*sin(theta)*sin(phi);
               z = r*cos(theta);
               atomPos(n,:) = [atomPos(n-1,1)+x,↩
                  atomPos(n-1,2)+y,atomPos(n-1,3)+z];
               safe = 1;
               if(atomPos(n,1) > box(1,2) || atomPos↩
                  (n,2) > box(2,2) || atomPos(n,3) > ↩
                  box(3,2) || atomPos(n,1) < box(1,1)↩
                   || atomPos(n,2) < box(2,1) || ↩
                  atomPos(n,3) < box(3,1) || sqrt(x↩
                  ^2+y^2+z^2) > 1)
                   safe = 0;
               end
           end
       end
    end
end

%set type 2
if(atomTypes == 2)
    atomTypeMatrix(atoms/2+1:atoms) = 2;
    atomChargeMatrix(1:atoms/2) = 12.08;
    atomChargeMatrix(atoms/2+1:atoms) = -12.08;
    %atomMomentMatrix(atoms,:) = [0,0,0];
    atomDiameterMatrix(1:atoms/2) = 1.0;
    moleculeMatrix(atoms) = moleculeMatrix(atoms-1)↩
       +1;
end

%put all matrices as a column in one large matrix ↩
   called atomMatrix
atomMatrix = [atomIdMatrix atomTypeMatrix atomPos ↩
   atomDiameterMatrix atomDensityMatrix ↩
   atomChargeMatrix atomMomentMatrix moleculeMatrix];


%generate file
fileID = fopen(['IonicLiquids/SolvationPolymer3A7C/↩
   atoms' num2str(m1*10,'%02d') 'D1E.txt'],'w');
fprintf(fileID,'# Model for \n\n');
fprintf(fileID,'%d\tatoms\n', atoms);
fprintf(fileID,'%d\tbonds\n\n', bonds);
fprintf(fileID,'%d\tatom types\n', atomTypes);
fprintf(fileID,'%d\tbond types\n\n', bondTypes);
fprintf(fileID,'%f\t%f\txlo xhi\n',box(1,1),box(1,2)↩
   );
fprintf(fileID,'%f\t%f\tylo yhi\n',box(2,1),box(2,2)↩
   );
```

```
    fprintf(fileID,'%f\t%f\tzlo zhi\n',box(3,1),box(3,2)↩
       );
    fprintf(fileID,'\n\nMasses\n\n');
    fprintf(fileID,'%d\t%f\n%d\t%f\n',1,masses(1,1),2,↩
       masses(1,2));
    fprintf(fileID,'\n\nAtoms\n\n');
    for i = 1:atoms
        fprintf(fileID,'%d\t%d\t%d\t%f\t%f\t%f\t%f\t%f\t↩
           %f\t%f\t%f\t%d\n',atomMatrix(i,1),atomMatrix(i↩
           ,2),atomMatrix(i,3),atomMatrix(i,4),atomMatrix↩
           (i,5),atomMatrix(i,6),atomMatrix(i,7),↩
           atomMatrix(i,8),atomMatrix(i,9),atomMatrix(i↩
           ,10),atomMatrix(i,11),atomMatrix(i,12));
    end
    if(bonds > 0)
        fprintf(fileID,'\n\nBonds\n\n');
        for i = 1:bonds
            fprintf(fileID,'%d\t%d\t%d\t%d\n',bondMatrix↩
               (i,1),bondMatrix(i,2),bondMatrix(i,3),↩
               bondMatrix(i,4));
        end
    end
    fclose(fileID);
end
```

Appendix D

Atom Generation Script Ver. 2

This version of the atom generation script was also written in matlab and used to generate the atom data files for the 3rd system mentioned in Chapter 4. It is generalized from the code given in Appendix C to allow for differing properties of particles to be used without changing the code too much. The code can be accessed and downloaded from `https://github.com/Shockersify/CreateParticles`.

D.1 CreateAtoms.m

```
% CreateAtoms.m
% Author: Cameron Shock
% Description: Create atom data files for LAMMPS. Use ↩
   the myscript.m file
% or your own as an interface with this function. ↩
   Parameters required and
% properties are structs and filename is a string.
% required must contain: box, masses, polymers, ↩
   numOfPolymers, bondTypes.
% properties contains the properties of each particle ↩
   type.

function atoms = CreateAtoms(required,properties,↩
   filename)

    % MAIN PART OF CODE DO NOT TOUCH↩
       ********************************

    bondType = 1;
    atomTypes = max(cell2mat(required.polymers));
    totalcolumns = numel(fieldnames(properties)) + 2;
    if any(strcmp(fieldnames(properties),'position'))
        totalcolumns = totalcolumns + 2;
    end
    if any(strcmp(fieldnames(properties),'moment'))
        totalcolumns = totalcolumns + 2;
    end

    % start particle id sequence
    atomMatrix = [];
    bondMatrix = [];
    id = 1;
    lastID = 0;
    offset = 0;

    % for each polymer chain
```

```
for polymer = 1:numel(required.polymers)
    % for the total number of this particular ↩
       polymer chain
    for n = 1:required.numOfPolymers(polymer)
        % initialize matrix holding polymer group ↩
           info
        polymerGroupInfo = ones(numel(cell2mat(↩
           required.polymers(polymer))),totalcolumns)↩
           ;
        % for each element in this particular ↩
           polymer chain
        for element = 1:numel(cell2mat(required.↩
           polymers(polymer)))
            types = cell2mat(required.polymers(↩
               polymer));

            %polymerGroupInfo(element,:) = [id, ↩
               types(element)];
            info = [id, types(element)];
            propertyFields = (fieldnames(properties)↩
               );
            propertyValues = struct2cell(properties)↩
               ;
            momentPosition = 0;
            prop = 2;
            for property = 1:numel(propertyFields)
                prop = prop + 1;
                pVal = cell2mat(propertyValues(↩
                   property));
                thing = pVal(types(element));
                if isequal(cell2mat(propertyFields(↩
                   property)), 'position')
                    thing = [1,1,1];
                    positionPosition = prop;
                    prop = prop + 2;
                end
                if isequal(cell2mat(propertyFields(↩
                   property)), 'moment')
                    thing = [1,1,1];
                    momentPosition = prop;
                    prop = prop + 2;
                end
                if isequal(cell2mat(propertyFields(↩
                   property)), 'molecule')
                    thing = n+offset;
                end
                info = [info thing];

            end

            % create the polymer group info matrix ↩
               holding all information
            % about each particle
            polymerGroupInfo(element,:) = info;
            id = id + 1;
        end

        % randomly generate positions and moments ↩
           then create bond
        % matrix
        if isfield(properties,'diameter')
            positions = MakePositions(↩
               polymerGroupInfo(:,1:2), required.box,↩
                properties.diameter);
        else
            positions = MakePositions(↩
               polymerGroupInfo(:,1:2), required.box,↩
                ones(atomTypes,1));
            end
            polymerGroupInfo(:,positionPosition:↩
               positionPosition+2) = positions;
            if momentPosition ~= 0
                moments = MakeMoments(polymerGroupInfo↩
                   (:,1:2), properties.moment);
                polymerGroupInfo(:,momentPosition:↩
                   momentPosition+2) = moments;
            end
            bonds = MakeBonds(polymerGroupInfo(:,1), ↩
               bondType, lastID);
            if size(bonds,1) > 0
                lastID = bonds(end,1);
            end

            % replace initialization data in polymer ↩
               group info with actual

            atomMatrix = [atomMatrix;polymerGroupInfo];
            bondMatrix = [bondMatrix;bonds];
        end
        offset = offset + required.numOfPolymers(polymer↩
           );
    end

    MakeFile(atomMatrix,bondMatrix,atomTypes,bondType,↩
       required.masses,required.box,filename);

    % END OF UNTOUCHABLE CODE↩
       ***************************************

end
```

D.2 MakePositions.m

```
% MakePositions.m
% Author: Cameron Shock
% Description: Given a matrix containg atom id's and ↩
   types, a matrix of box
% size, and a matrix of particle diameters randomly ↩
   place the first
% particle of a polymer chain and use random walk to ↩
   create the rest.

function positions = MakePositions(atoms, box, diameter)
    positions = ones(size(atoms,1),3);
    posX = box(1,2) * rand();
    posY = box(2,2) * rand();
    posZ = box(3,2) * rand();
    positions(1,:) = [posX, posY, posZ];
    for x = 1:size(atoms,1)
        % for first particle pick random x,y,z
        if x == 1
            posX = box(1,2) * rand();
            posY = box(2,2) * rand();
            posZ = box(3,2) * rand();
        % for subsequent particles use 3D random walk
        else
            oldX = posX;
            oldY = posY;
            oldZ = posZ;
            while 1
                r = diameter(atoms(x,2));
                phi = 2*pi*rand();
                theta = pi*rand();
                posX = posX + r*sin(theta)*cos(phi);
                posY = posY + r*sin(theta)*sin(phi);
                posZ = posZ + r*cos(theta);


                % check that new position is within box
                if(posX > box(1,1) && posY > box(2,1) &&↩
                    posZ > box(3,1) && posX < box(1,2) &&↩
                    posY < box(2,2) && posZ < box(3,2))
                    break
                % if not reset position and try again
                else
                    posX = oldX;
                    posY = oldY;
                    posZ = oldZ;
                end
            end
        end
        positions(x,:) = [posX, posY, posZ];
    end
end
```

D.3 MakeMoments.m

```
% MakeMoments.m
% Author: Cameron Shock
% Description: Given a matrix of atom id's and types and↩
    a matrix of the
```

```
% dipole moments for each type, create a matrix giving ↩
   random dipole moment
% directions to the atoms.

function moments = MakeMoments(atoms, moment)
    moments = ones(size(atoms,1),3);
    for p = 1:size(atoms,1)
       phi = 2*pi*rand();
       theta = pi*rand();
       x = moment(atoms(p,2))*sin(theta)*cos(phi);
       y = moment(atoms(p,2))*sin(theta)*sin(phi);
       z = moment(atoms(p,2))*cos(theta);
       moments(p,:) = [x,y,z];
    end
end
```

D.4 MakeBonds.m

```
% MakeBonds.m
% Author: Cameron Shock
% Description: For a given matrix of atom id's, int of ↩
   bondType, and int of
% last particle id, create a matrix of the bond ↩
   connection info for given
% atoms.

function bonds = MakeBonds(atoms, bondType, lastID)
    bonds = [];
    if size(atoms,1) > 1
        bonds = ones(size(atoms,1)-1,4);
        for x = 1:size(atoms,1)-1
            bonds(x,:) = [x+lastID, bondType, atoms(x), ↩
               atoms(x+1)];
        end
    end
end
```

D.5 MakeFile.m

```
% MakeFile.m
% Author: Cameron Shock
% Description: Given all of the matrices containg the ↩
   atom information and
% all other info (should hopefully be obvious) create ↩
   the atom data file
% for LAMMPS.

function file = MakeFile(atomMatrix, bondMatrix, ↩
   atomTypes, bondTypes, masses, box, filename)
    %generate file
    fileID = fopen(filename,'w');
    fprintf(fileID,'# Model for \n\n');
    fprintf(fileID,'%d\tatoms\n', atomMatrix(end,1));
    if(size(bondMatrix,1) > 0)
        fprintf(fileID,'%d\tbonds\n\n', bondMatrix(end↩
           ,1));
    else
        fprintf(fileID,'%d\tbonds\n\n', 0);
    end
    fprintf(fileID,'%d\tatom types\n', atomTypes);
    fprintf(fileID,'%d\tbond types\n\n', bondTypes);
    fprintf(fileID,'%f\t%f\txlo xhi\n',box(1,1),box(1,2)↩
       );
    fprintf(fileID,'%f\t%f\tylo yhi\n',box(2,1),box(2,2)↩
       );
    fprintf(fileID,'%f\t%f\tzlo zhi\n',box(3,1),box(3,2)↩
       );
    fprintf(fileID,'\n\nMasses\n\n');
    for i = 1:size(masses,2)
        fprintf(fileID,'%d\t%f\n',i,masses(i));
    end
    fprintf(fileID,'\n\nAtoms\n\n');
    % for each atom
    [m,n] = size(atomMatrix);
    for i = 1:m
        for j = 1:n
            fprintf(fileID,'%f\t',atomMatrix(i,j));
        end
        fprintf(fileID,'\n');
    end
    % if bonds exist create the bonds section for each ↩
       bond
    if(size(bondMatrix,1) > 0)
        fprintf(fileID,'\n\nBonds\n\n');
        for i = 1:bondMatrix(end,1)
            fprintf(fileID,'%d\t%d\t%d\t%d\n',bondMatrix↩
               (i,1),bondMatrix(i,2),bondMatrix(i,3),↩
               bondMatrix(i,4));
        end
    end
    fclose(fileID);
end
```

D.6 myscript.m

```
% myscript.m
% Author: Cameron Shock
% Description: Create atom data files for LAMMPS. First ↩
   construct the
% polymer cell matrix which consists of arrays of ↩
   whatever atom types you
% want. Next numOfPolymers determines the number of each↩
    polymer chain you
% want. The next section is defining all of properties ↩
   for each atom type
% (each column for each type).


% set size of the simulation box
box = [0.0000,10.0000;0.0000,10.0000;0.0000,10.0000]; %↩
   box dimensions

% info based on polymer sets
polymers = {{[1],[2]}}; % each matrix is a different ↩
   polymer chain, each element of a matrix is the atom ↩
   type of that particle
numOfPolymers = [278,1]; %number of each polymer chain
bondTypes = {[1]}; %ignore for now

for n = 1:numel(ions(:,1))

    name = cell2mat(ions1(n));
    if (name(end) == '-')
        charge = -2;
    else
        charge = 2;
    end

    % info based on atom types (each element is value ↩
       for that type)
    charge = [0,charge * 12.08];
    moment = [1.391621,0];
    diameter = [0.7666,ions(n,1)];
    density = [1,1];
    masses = [46.1,ions(n,2)];

    % do not touch
    position = ones(numel(masses));
    molecule = ones(numel(masses));
    required = struct('box',box,'masses',masses,'↩
       polymers',polymers,'numOfPolymers',numOfPolymers,'↩
       bondTypes',bondTypes);
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


    %ion = cell2mat(ions1(i));
    filename = ['../Sandia/SolvationExperimental/Eth2.3/↩
       atomsEth' cell2mat(ions1(n)) '.txt'];

    % order and name of properties needed for this atom ↩
       type
    properties = struct('position',position,'diameter',↩
       diameter,'density',density,'charge',charge,'moment↩
       ',moment,'molecule',molecule);

    CreateAtoms(required,properties,filename);

end
```

Appendix E

Non-Polymer Example Script

E.1 scriptSolvMono10ps400k_5

```
#####################################################
#                                                   #
#                                                   #
# Filename: scriptSolvMono10ps400k_5        #
# Author: Cameron Shock, 2018                       #
#               #
# Execute the script through:                       #
# lmp_exe < script.in                       #
#                                                   #
#####################################################

# VARIABLES
```

```
variable fname index atoms25D-1E.txt
variable simname index 25D-1E

# Initialization
units   lj
boundary  p p p
atom_style  hybrid sphere dipole bond
log     log.${simname}.txt
read_data ${fname}

# Dreiding potential information
neighbor  0.4 bin
neigh_modify  every 10 one 10000

#####################################################
#Minimization
pair_style  soft 1.225
pair_coeff  * * 60.0
velocity  all create 1.048 1231
timestep  0.00187 #2fs
min_style cg

minimize 0.0 1.0e-10 1000000 1000000
####################################################
# Equilibration
pair_style  lj/cut/dipole/long 1.225 5.0
pair_coeff  1 1 1.0 1.0
pair_coeff  2 2 1.0 0.666 0.748
pair_modify mix geometric
kspace_style  ewald/disp 1e-4
special_bonds fene

#Collect MSD to calculate Diffusion Coefficient
group 2 type 2
```

```
compute myMSD 2 msd
fix 3 all ave/time 200 10 2000 c_myMSD[*] file ${simname↩
   }.msd mode vector


dump mf1 all custom 100 ${simname}.dump id type xs ys zs↩
    mux muy muz q

velocity  all create 1.048 1231
fix       1 all nve/sphere update dipole
fix       2 all langevin 1.048 1.048 1.0 573456 omega ↩
   yes zero yes
thermo_style  custom step temp etotal pe epair
thermo          5000
timestep  0.00187 #2fs
run   5100000

unfix 1
unfix 2
unfix 3



print "All done"
```

Appendix F

Polymer Example Script

F.1 scriptSolvMonoP16_1

```
#####################################################
#                                                   #
#                                                   #
# Filename: script1D1E2.in                #
# Author: Cameron Shock, 2018                       #
#               #
# Execute the script through:                       #
# lmp_exe < script.in                       #
#                                                   #
#####################################################

# VARIABLES
```

```
variable fname index atoms05D-1E16P.txt
variable simname index 05D-1E16P

# Initialization
units   lj
boundary  p p p
atom_style  hybrid sphere dipole bond
log     log.${simname}.txt
read_data ${fname}

# Dreiding potential information
neighbor  0.4 bin
neigh_modify  every 10 one 10000

#####################################################
#Minimization
pair_style  soft 1.68369
pair_coeff  * * 60.0
bond_style  fene
bond_coeff  * 15.556 2.25 1.0 1.5
special_bonds fene
velocity  all create 1.048 1231
timestep  0.0028 #2fs
min_style cg

minimize 0.0 1.0e-10 1000000 1000000
####################################################
# Equilibration
pair_style  lj/cut/dipole/long 1.68369 7.5
pair_coeff  1 1 1.0 1.5
pair_coeff  2 2 1.0 1.0 1.225
pair_modify mix geometric
kspace_style  ewald/disp 1e-4
bond_style  fene
```

```
bond_coeff  * 15.556 2.25 1.0 1.5
special_bonds fene

#Collect MSD to calculate Diffusion Coefficient
group 2 type 2
compute myMSD 2 msd
fix 3 all ave/time 200 10 2000 c_myMSD[*] file ${simname↩
   }.msd mode vector


dump mf1 all custom 5000 ${simname}.dump id type xs ys ↩
   zs mux muy muz q

velocity  all create 1.048 1231
fix       1 all nve/sphere update dipole
fix       2 all langevin 1.048 1.048 1.0 573456 omega ↩
   yes zero yes
thermo_style  custom step temp etotal pe epair
thermo          5000
timestep  0.0028 #2fs
run   51000000

unfix 1
unfix 2
unfix 3




print "All done"
```

Appendix G

Experimental Replication Example Script

G.1 scriptSolvLi+10ps400k

```
#####################################################
#                                                   #
#                                                   #
# Filename: scriptSolvLi+10ps400k.in        #
# Author: Cameron Shock, 2019                       #
#              #
# Execute the script through:                       #
# lmp_exe < scriptSolvLi+10ps400k.in        #
#              #
# Purpose: Simulate a Li+ atom in a solvent of      #
# methanol molecules using Stockmayer Fluid         #
# model.              #
#                                                   #
#####################################################

# VARIABLES
variable fname index atomsMethLi+.txt
variable simname index MethLi+

# Initialization
units   lj
boundary  p p p
atom_style  hybrid sphere dipole bond
log     log.${simname}.txt
read_data ${fname}

# Dreiding potential information
neighbor  0.4 bin
neigh_modify  every 10 one 10000

######################################################
#Minimization
pair_style  soft 1.225
pair_coeff  * * 60.0
velocity  all create 1.048 1231
timestep  0.0021 #2fs
min_style cg

minimize 0.0 1.0e-10 1000000 1000000
#####################################################
# Equilibration
pair_style  lj/cut/dipole/long 1.225 5.0
```

```
pair_coeff  1 1 1.0 0.8666 0.973  # methanol 2.6 ↩
   angstrom
pair_coeff  2 2 1.0 0.5200 0.5837 # lithium 0.78 ↩
   angstrom
pair_modify mix geometric
kspace_style  ewald/disp 1e-4
special_bonds fene

#Collect MSD to calculate Diffusion Coefficient
group 2 type 2
compute myMSD 2 msd
fix 3 all ave/time 200 10 2000 c_myMSD[*] file ${simname↩
   }.msd mode vector


dump mf1 all custom 5000 ${simname}.dump id type xs ys ↩
   zs mux muy muz q

velocity  all create 1.048 1231 # 400k
fix       1 all nve/sphere update dipole
fix       2 all langevin 1.048 1.048 1.0 573456 omega ↩
   yes zero yes
thermo_style  custom step temp etotal pe epair
thermo          5000
timestep  0.0021 #2fs
run   51000000

unfix 1
unfix 2
unfix 3



print "All done"
```