

EXPLORING CRITICAL CONFORMATIONS:
STATE SEARCHING AND SAMPLING IN BOTH
GERMANIUM CHAINS AND ICE INTERFACES
{EARLY DRAFT}

By

GENTRY H. SMITH

B.S.

Southern Nazarene University

Bethany, OK, USA

2016

Submitted to the Faculty of the
Graduate College of
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
Master of Science
November 2018

EXPLORING CRITICAL CONFORMATIONS:
STATE SEARCHING AND SAMPLING IN BOTH
GERMANIUM CHAINS AND ICE INTERFACES
{EARLY DRAFT}

Thesis Approved:

Thesis Advisor

ACKNOWLEDGMENTS

Acknowledgments reflect the views of the author and are not endorsed by committee members or Oklahoma State University.

ACKNOWLEDGMENTS

To Oklahoma State University, for providing the environment in which I have been able to study, teach, and research.

To the HPCC and the individuals within (**get first/last names**) for providing a powerful cluster for computations and continuous support for technical issues.

To my advisor, who instructed and assisted me in research

To my parents, by blood and marriage, who have always encouraged me toward higher goals.

To my wife, Miranda, who has supported me for over 5 years.

Acknowledgments reflect the views of the author and are not endorsed by committee members or Oklahoma State University.

Name: GENTRY H SMITH

Date of Degree: November 2018

Title of Study: EXPLORING CRITICAL CONFORMATIONS

Major Field: COMPUTATIONAL CHEMISTRY

Abstract: Molecular conformation plays a critical role in the properties of systems in either the condensed or vapor states. The ensemble of conformations dictates structural properties, energies, heat capacities, and other thermodynamic and dynamic quantities. Here, we explore the role of conformation in proton ordering and orientational defect formation in ice as well as strategies for exhaustive conformer searching for molecules using Group IV element backbones. In the ice systems, we show algorithmic strategies for seeking optimized proton disordered crystals that satisfy the Bernal-Fowler ice rules. In the Group IV molecule investigations, we develop an automated strategy for seeking the optimal low energy conformer and uncover previously unreported deficiencies in common computational software used in investigating Germanium complex energies.

TABLE OF CONTENTS

Chapter		Page
1	Introduction	1
1.1	First Words	1
1.2	A Brief History of Computational Chemistry	1
1.3	Reason for Study	2
1.3.1	Generation of Ice I_h Crystal Structure	2
1.3.2	Conformation Landscapes of Group IV Chains	2
2	On Algorithms for Building and Sampling Disordered Crystal States	3
2.1	States and Properties of Ice	3
2.1.1	Bernal-Fowler Ice Rules	3
2.1.2	Forms of Ice	3
2.1.3	Ice I_h	4
2.1.4	Efforts to Generate Ice I_h	4
2.1.5	Comparison between Ice XI and Ice I_h	4
2.2	Method Design	5
2.2.1	Overview	5
2.2.2	Selection of Software Tools	5
2.2.3	Generation of Source Ice XI	5
2.2.4	Source Ingestion	5
2.2.5	Identifying Neighboring Molecules	6
2.2.6	Defining Tetrahedral Positions	6
2.2.7	Pseudorandom Rearrangement of Water Molecules	7
2.2.8	Detecting Hydrogen Bond Defects	8
2.2.9	Correcting Hydrogen Bond Defects	8
2.3	Results of Method	9
2.4	Comparison with Other Methods	11
2.4.1	benefits of own method over others	11
2.4.2	benefits of other methods over this	11
2.5	Comments on Limitations and Proposed Improvements	11
3	Crystal and Liquid 2D Water at Interfaces	12
3.1	Two-Dimensional Water	12
3.1.1	Lennard-Jones Potential	12
3.1.2	Modeling Water in Two Dimensions	13
3.1.2.1	Mercedes-Benz Model	13
3.1.2.2	Rose Potential Model	14

3.1.2.3	Two-Dimensional Modeling	14
3.2	Goal of Project	15
3.3	Tools and Terms	15
3.3.1	OOPSE in 2D	15
3.3.2	Reduced Terms: 2D analogues	15
3.4	Designing System	15
3.4.1	Defining the Surface	16
3.4.1.1	Manipulation of LJ Potential	16
3.4.1.2	Manipulation of bead spacing	16
3.5	Results	16
4	Germanium Compounds and QM Concerns	17
4.1	Modeling Germanium Compounds	17
4.1.1	Computational Complexity of Germanium Compounds	17
4.1.2	Literature Review on Relevant Works	17
4.2	The Initial Problem: Germanium Study	17
4.2.1	Parameters of Work and Previous Collaborator's Results	18
4.2.2	Design and Approach to Solution	21
4.2.2.1	Design 1: Occam's Smallest Razer	21
4.2.2.2	Design 2: A Blunt Effort	21
4.2.2.3	Design 3: Death by 1.59 Million Cuts	25
4.2.3	Scale Reduction Efforts	27
4.2.4	Efforts at Simplification	28
4.2.5	Discovery of a Consistent Inconsistency	32
5	Sampling Conformation Landscapes by Rotatable Bond Degrees of Freedom	33
5.1	Introduction to Topic	33
5.1.1	A Brief History on Conformation Landscapes	33
5.1.1.1	Levinthal's Paradox	33
5.1.2	Literature Review on Relevant Works	33
5.2	Design of System	34
5.2.1	Variation of Theory and Basis Set Usage by System Size and largest atom type	36
5.2.2	Computational Optimization by Varying Resolution	36
5.2.3	Inherent Complications	36
5.3	Results	36
5.3.1	Problems	36
5.3.2	Anticipated Approaches for Future Work	37
	References	38
A	Ice Ih to Ice XI Conversion	39
A.1	Code: Crystal Disorganizer Tool	39

B	Two-Dimensional Rose-Potential Water	51
B.1	Code: Surface Adjustment Tool	51
C	Germanium Landscape	55
C.1	Sample Gaussian 09 Germanium File	55
C.2	Building Group 4 Chains	58
C.3	Collecting and Comparing Torsional Data	60
D	Conformation Landscapes	67
D.1	Code: hexagermane-transall.pdb	67
D.2	Code: ge4h.pdb	76
D.3	Progress on Torsion Minimizer System	77

LIST OF TABLES

Table		Page
4.1	Collaborator's Hexagermanium Energies by Conformation (density functional theory, unknown basis set, energy in Hartrees and KJ/mol) . .	19
4.2	Data of B2LYP/STO-3G minimization of variations of pentagermane compound at various conformers. DNC denotes a failure to converge with the self-consistent field method.	23
4.3	Data of B2LYP/STO-3G minimization of variations of hexagermane compound at various conformers. DNC denotes a failure to converge with the self-consistent field method.	24

LIST OF FIGURES

Figure	Page
2.1 "Before" image of Ice XI	10
2.2 "After" image of generated ice Ih	10
4.1 Fully trans configuration of pentagermanium-based compound.	18
4.2 Fully trans configuration of hexagermanium-based compound.	19
4.3 Sample Newman projection of cis-butane.	20
4.4 Visualization of a trans-cis-trans hexagermane structure.	28
4.5 Visualization of a multiple pure group IV torsions at various theories and basis sets	29
4.6 Visualization of a multiple pure group IV torsions at various theories and basis sets	31
5.1 example VR chart (hand-drafted, CGI pending)	34
5.2 Flow of method design for variable resolution conformation landscape search.	35

NOMENCLATURE

Variables

ϵ	Lennard-Jones Potential well depth
r	Lennard-Jones Potential distance between center of two particles
σ	Lennard-Jones Potential intermolecular contact distance
V	Lennard-Jones Potential intermolecular potential

Subscripts/Superscripts

0	Initial condition
---------	-------------------

CHAPTER 1

Introduction

1.1 First Words

For nearly a century, computational chemistry has greatly assisted wet lab research and discovery and has relatively recently become its own field of focus within chemistry.

1.2 A Brief History of Computational Chemistry

The section is a work in progress and will be expanded upon during the next week.

I want a first paragraph that inspires the reader to continue reading. Maybe something with a quote or a question. Maybe not.

The idea for an introduction is to interest the reader and provide general background information. For my work, the interesting part is how impactful computational science (specifically chemistry) has been in research and on society. The background information will start extremely generic and then go into some overarching themes. Generically, I'll include the development of computational science in the 20th century and the vast applications of computational chemistry specifically. Overarching themes are related, but don't have to be explicitly relevant. For example, Levinthal's paradox is a fun example showing the problem of conformation landscape searches. Also,

1.3 Reason for Study

The studies conducted that comprise this work were determined as a combination of collaborative efforts and larger research group goals with results and discoveries worth reporting. Collectively, they explore conformations of internal bond dihedrals and molecule orientations of microstates. These efforts are categorized and separated into sections. A brief introduction of each and a literature review of relevant information is given below.

1.3.1 Generation of Ice I_h Crystal Structure

Ice crystals can take many forms based on properties like temperature and pressure. The proton-disorganized orthorhombic form of ice known as I_h is the form of ice most commonly found on earth (general understanding, CITATION NEEDED). Due to the inherent randomness of the disorganization of the molecules within the crystal, computational efforts are limited in scope or instead utilize the proton-ordered orthorhombic form of ice XI (general understanding, CITATION NEEDED). This project explores a method to produce a high quality pseudorandom ice I_h crystal structure.

1.3.2 Conformation Landscapes of Group IV Chains

Any molecule with a chain length of at least four contains at least one dihedral. In small molecules, the steric hindrance between the head and tail atoms are usually minimized in the fully gauche conformation to produce the lowest-energy conformer. In larger and more bulky molecules, additional interactions may cause the dihedral to take other conformations in search of the lowest-energy conformation. This project details the search for the lowest-energy conformer of a bulky hexagermane molecule in collaboration with Oklahoma State University’s Charles Weinert and the complications and curiosities found within.

CHAPTER 2

On Algorithms for Building and Sampling Disordered Crystal States

2.1 States and Properties of Ice

Ice is cool. Ice has many forms, each with unique environments and structures that give rise to similar and unique properties.

2.1.1 Bernal-Fowler Ice Rules

Bernal-Fowler Ice Rules are the basic rules for how water molecules interact in an ice structure. DETAILS ON BF PAPER

Basically, water's tetrahedral structure allows for four interactions on each molecule. The two protons allow for a hydrogen bond with a lone pair from a neighboring oxygen atom. Similarly, the oxygen atom's two lone pairs allow for a hydrogen bond with a neighboring proton. These rules are fairly rigid in the sense that every water molecule can interact with two oxygen atoms and two protons from four surrounding water molecules. These are also free-form in the sense that each of the four attached water molecules can occupy one of three rotational microstates, allowing for 81 possible configurations (including rotational duplicates).

2.1.2 Forms of Ice

While ubiquitous in the 'I_h' form, ice water has many known forms. As of the writing of this work, there are 17 established forms of ice. These forms usually occur in cubic, hexagonal, and orthorhombic crystal structures. The relationship between external pressure and temperature are the primary defining characteristics of which form will

form in a given system. Do other characteristics come into play???????

2.1.3 Ice I_h

As the most commonly found form on earth, ice I_h is a highly desired form for computational studies involving ice systems. As an interesting specificity, ice I_h does not cover any proton-disordered crystal with the same oxygen spatial positioning. Rather, it holds a specific unit cell defined in (source?).

2.1.4 Efforts to Generate Ice I_h

Cover Buch et al. (1998) work here and additional discoveries.

2.1.5 Comparison between Ice XI and Ice I_h

While ice I_h is known as the most common form of ice found on the planet, it is much more difficult to computationally generate than an ice XI crystal. The ease of generation of an ice XI structure stems from the repetition of a unit cell with consistent layering and orientation throughout the crystal lattice.

With ice I_h crystals, the proton-disordered form introduces entropy by way of rotational disorder. As the protons and lone pairs are no longer consistently ordered, hydrogen bonds may no longer form properly at all interaction sites. The interaction of proton with proton or lone pair with lone pair are not hydrogen bonds and are considered defects in the lattice. An ice structure of randomly oriented molecules without consideration of hydrogen bonds will likely produce defects at many interaction sites across the lattice and weaken the integrity of the system, leading to stability problems while running simulations. In generating the crystal, the cause of these defects must be considered and countered effectively.

2.2 Method Design

2.2.1 Overview

The big idea is to convert an easy-to-make ice XI crystal into an ice Ih crystal. Because the key difference in structure is the proton-orderedness, it might be possible to rearrange the water molecule orientations in a pseudorandom way to create an ice Ih crystal. This section walks through the method developed to convert ice XI into ice Ih, the results of initial testing, and imperfections discovered in the design.

2.2.2 Selection of Software Tools

Python was chosen as the language of the tool due to the versatility of the language and the ease of development due to the "pseudocode" written style of the language and the availability of scientific packages including SciPy and NumPy. Python version 2.7 was specifically chosen due to familiarity with the language. Crystal files were defined and saved as Protein Data Bank (.pdb) files as this format allows for defining multiple molecules within a larger structure with a simple X, Y, Z grid position format.

2.2.3 Generation of Source Ice XI

This is Dr. Fennell's method to create an ice XI pdb file. Basically, the ice XI unit cell of eight water molecules is repeated as desired to create a sufficiently large crystal. The primarily used crystal consists of a 3 x 3 x 6 unit cell repetition totaling 432 water molecules.

2.2.4 Source Ingestion

It is important that the crystal be read and stored in an efficient method to keep relevant information about each molecule easily accessible. As the file is read in, each

molecule is stored as an entry in a multidimensional array where the first index is the molecule number. Further, the second index defines the molecule number where 0 is oxygen and 1 and 2 are the protons. The third, fourth, and fifth indices define the X, Y, and Z position coordinates.

2.2.5 Identifying Neighboring Molecules

Identifying the neighboring molecules proved computationally difficult. The most effective method is to find the closest four molecules by computing a distance calculation between every two oxygen atoms. This ensures every molecule is considered, but also presents significant hurdles. First, a distance calculation utilizes an extremely computationally-inefficient square root calculation, which can be ignored by instead calculating the squared-distance between molecules and finding the lowest values.

Second, molecules on the walls and edges of the molecule will not have four neighbors in the non-periodic crystal. This is accounted for by shifting all six sides to make a pseudo-periodicity for these edge cases. Those periodically-neighboring molecules are flagged with a shifting value in the neighboring atom array by specifying a translation in the x, y, or z axis values. Unfortunately, the necessary code to implement the periodically-neighboring molecule detections requires a major rewrite of the entire tool and has not yet been implemented.

Once these closest neighboring oxygen atoms have been discovered, the appropriate interacting tetrahedral position is identified by finding the closest of the four tetrahedral positions using the same squared-distance calculation with the four defined tetrahedral positions detailed in the next subsection.

2.2.6 Defining Tetrahedral Positions

An important aspect of pseudorandom selection is the existence of a bank of options. Using the ingestion portion to calculate and store all tetrahedral possibilities proves

useful. For each water molecule, the first two tetrahedral positions are known by the positions of the two hydrogen atoms. The other two positions are found by rotating one hydrogen atom 120° twice about the vector from the oxygen atom through the other hydrogen atom and storing the resulting positions as tetrahedral positions three and four.

This does not produce an exactly correct tetrahedral position of potential hydrogen atoms due to the slight acuteness of the H-O-H bond created by the variance in repulsive forces between the two lone pairs of electrons and two hydrogen atoms. Fortunately, this difference is sufficiently small for visualization programs like Avogadro to still recognize hydrogen bonds between a rotated hydrogen atom and corresponding neighboring lone pair. Currently, the method relies on a very soft annealing process by a simulation package to minimize the effect of this hydrogen bond imperfection. Future versions of this method may account for the variations.

2.2.7 Pseudorandom Rearrangement of Water Molecules

Once the tetrahedral positions have been defined, each water molecule is ready to rotate. What may seem the most crucial step in this methods ends up being the most simple. As designed, the rotation of water molecules is as simple as using a stepwise iterator to pseudorandomly select two tetrahedral positions for the hydrogen bonds and store the new positions in a new crystal array. An extremely important note is that this rearrangement does not consider the orientations of neighboring molecules and likely introduces defects of hydrogen - hydrogen and lone pair - lone pair interactions. The likelihood of a defect-free interaction lattice forming is nearly zero and is assumed to have a great deal of defects within the lattice.

2.2.8 Detecting Hydrogen Bond Defects

After all water molecules have been rearranged, defects between incorrectly-interacting hydrogen bonds must be found and corrected. Discovering the defects relies on the detection of neighboring molecules and the appropriate interacting hydrogen atom or electron lone pair. As previously discussed, the initial data ingest records and detects the nearest water molecules and determines the tetrahedral position containing the interacting space, be it electron lone pair or hydrogen atom. From that data, the detection of a valid hydrogen bond is as simple as checking both all interacting tetrahedral positions and confirming that they both do not contain or lack a hydrogen atom. Additionally, each water molecule keeps a count of how many defects are present among the four positions. This allows for contextual changes during the correction step.

2.2.9 Correcting Hydrogen Bond Defects

Once the hydrogen bond defects have been discovered and marked, each needs to be corrected. The most direct approach to this is to sequentially walk through each defect and repeat the pseudorandom rotation until the number of defective regions is zero or a user-specified value. The current implementation sorts the defect list by the number of defects and attempts to fix the most defective molecules first. The most defective molecules may include defects impossible to solve by simple rotation, specifically when neighboring molecules have collectively directed three or four hydrogen atoms or electron lone pairs at the target water. These can only be solved by adjusting one or more of the neighboring molecules until the number of hydrogen atoms and electron lone pairs have balanced. Unfortunately, this high-defect problem can quickly escalate if the neighboring molecules contain the same problem of unbalanced hydrogen atoms and electron lone pairs. The current solution is to recursively check for and fix these impossible interactions first, but has not yet yielded a defect-free crystal in testing.

The current design of the method allows for the user to specify a threshold of defects as an average per molecule. For example, a threshold of 2.5 will allow a maximum of 3 defects on any given molecule and will continue to correct defects until the average number of defects per molecule is equal to or below 2.5. Because each of these defects will be counted twice, once for each molecule, the total number of defects in a crystal can be determined by multiplying the average defect value by the number of molecules and dividing by two. As of the current implementation, the method cannot reliably produce a crystal with a threshold below 2 as it will continue to search until the system runs out of available memory and crash without recording any new structure.

2.3 Results of Method

When supplied with an input ice XI crystal, an output structure with rotated water molecule orientations strictly consistent with ice Ih describes a success at the most basic level. An example before and after of the method is given in figures 2.1 and 2.2. As can be seen, the "after" image has experienced rotation and can no longer be classified as ice XI. However, as ice I_h also has a standard shape, the generated crystal can not be considered ice I_h . Instead, it can be considered a proton-disordered orthorhombic ice crystal similar to ice I_h .

Unfortunately, the result is not without defect. When following the subsequent layers in the crystal, patterns emerge. Inconsistently, some rows of waters remain consistent. Some of these are a uniform rotation of both hydrogen atoms, while others are just one consistently placed hydrogen atom. Multiple trials yield internally unique results, yet all contain these strange consistencies. This may be due to some accidental pattern in the method's implementation.

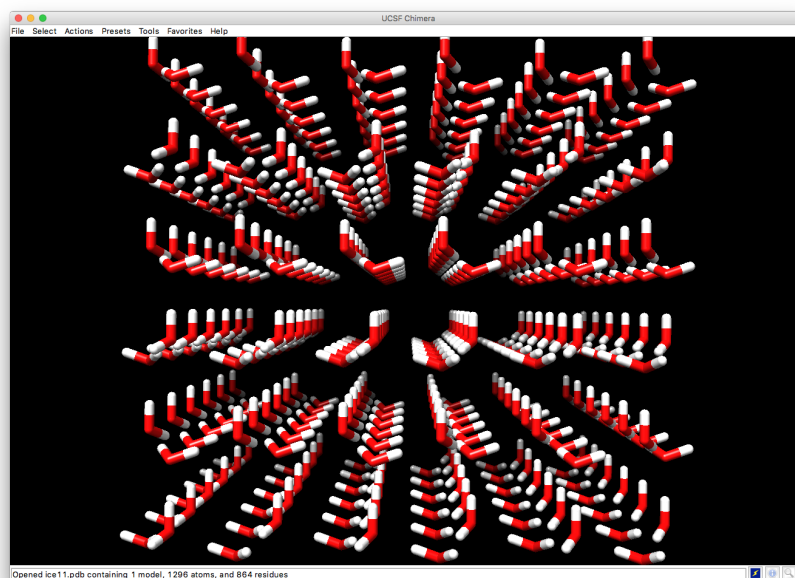


Figure 2.1: "Before" image of Ice XI

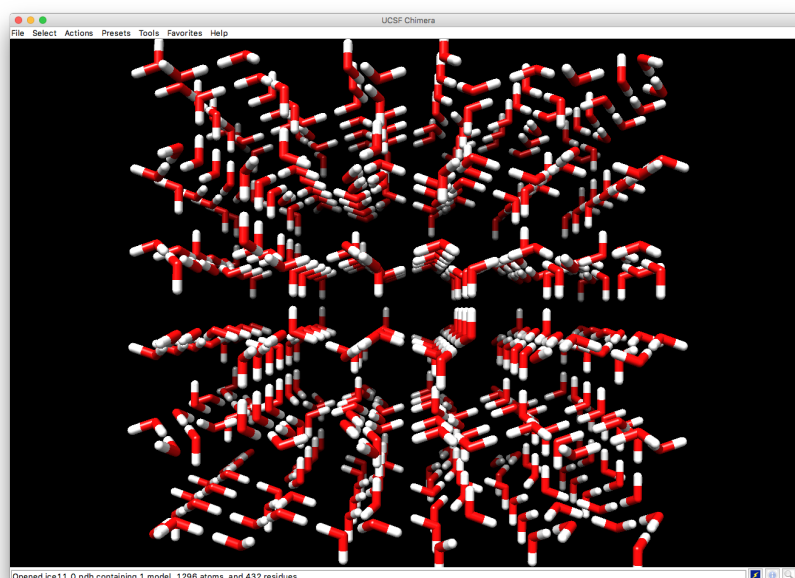


Figure 2.2: "After" image of generated ice Ih

2.4 Comparison with Other Methods

Currently in progress, this section will primarily compare the results of this method with Buch et al. (1998).

2.4.1 benefits of own method over others

2.4.2 benefits of other methods over this

2.5 Comments on Limitations and Proposed Improvements

During the hydrogen bond defect correction step, a weakness in the design is that any clustering or regions of high defect density will not be noticed. This allows the existence of a highly-defective region within the larger structure that could potentially cause problems when the crystal is used in simulations. The prevalence and occurrence of these defects have not been studied, but seem a natural inevitability of statistics. A potential solution with partial development will score regions based on the number of defects as a weighted function expanding out from a central molecule for N connections. For example, consider a given molecule defined as level 1. The neighboring four molecules are defined as level 2, and continued onward excepting already-defined molecules out to an N^{th} level. The number of defects in each level can be counted and averaged. Then a depressive factor along the lines of $\frac{1}{level}$ can be used to diminish the value of defects further away from the first-level molecule. This would create a value for each molecule that shows the relative density of defects centered about that specific molecule and could even be plotted as a gradient change within the crystal. The general approach to a scoring mechanism may take a form similar to equation 2.1.

$$Value = \sum_{l=1}^{N_{levels}} \left[\frac{1}{l} * \frac{1}{N_{molecules}} * \sum_{m=1}^{N_{molecules}} [N_{defects,m}] \right] \quad (2.1)$$

CHAPTER 3

Crystal and Liquid 2D Water at Interfaces

3.1 Two-Dimensional Water

EACH SUBSECTION: DEFINITION OF TERMS

Rose water is fairly new on the computational scene and so I may also include a review on the Mercedez-Benz water system as well as any other attempts to model water in two dimensions. For the rose potential system, I will review the Lennard-Jones potential as well as any other equations/systems related to the rose potential.

3.1.1 Lennard-Jones Potential

The Lennard-Jones Potential well is a soft-sphere model of interaction between two spheres described with

$$V_{LJ} = 4\epsilon[(\frac{\sigma}{r})^{12} - (\frac{\sigma}{r})^6] \quad (3.1)$$

where V is the potential, r is the distance between the center of two particles, σ is the specific distance between the two particles where the potential is zero, and $-\epsilon$ is the minimum potential of the plot. **REFINE WORDING:** The plot is defined in $[0, \infty)$. As two particles approach from infinity, their interaction become negative - which is an attractive force - and will approach the global minimum of $-\epsilon$. The r of this interaction is slightly larger than the combined radii of the two particles - which means they aren't quite touching - and is the equilibrium distance between the two particles. As r decreases beyond the minimum and toward σ , the interaction strength

increases and reaches zero as $r = \sigma$. At $r < \sigma$,

Potential digression: In a "hard-sphere" model, a particle's radius is firm, which is to say that the interaction potential is infinite at r less than σ . Basically a vertical line between two discrete values (usually ϵ and ∞) as the potential shifts from $r \geq \sigma$ to $r < \sigma$ (maybe include image?). The Lennard-Jones potential is a "soft-sphere" model, which blurs the line and replaces the vertical line with a functional representation. This breaks with reality as the particles become "squishy" and the potential ramps up toward infinity as r decreases. The benefit to the soft-sphere model is that modeling programs can more-easily account for overlaps in particles during time steps than with hard-sphere models. For example, a hard-sphere model of two particles interacting will likely not have a position where $r = \sigma$ and will potentially overlap. At this overlap, the potential is infinity and will introduce a nearly infinite force at that instant of time. Computer systems do not like having infinitely large repulsions suddenly introduced into a simulation.

3.1.2 Modeling Water in Two Dimensions

Modeling in two dimensions sacrifice the "realism" of models in three dimensions, but reduce the computational load significantly. This allows researchers (scientists, chemists, digital magicians?) to test more simple designs in two dimensions as well as a higher volume of simulations at the same time/computational cost.

3.1.2.1 Mercedes-Benz Model

The "Mercedes-Benz" BN2D model of water first proposed by Ben-Naim (1971) as "waterlike particles" are a popular two-dimensional representation of water. ROUGH: details of shape of MB water

The mathematical model used in the BN2D model is generated from the Percus-Yevick equation by substituting the approximation

$$c(X_1, X_2) = y(X_1, X_2)f(X_1, X_2) \quad (3.2)$$

into the Percus-Yevick equation obtained from the Ornstein-Zernike relation

$$h(X_1, X_2) = c(X_1, X_2) + \frac{\rho}{2\pi} \int c(X_1, X_3)h(X_3, X_2)dX_3 \quad (3.3)$$

to produce the overall relation

$$y(X_1, X_2) = 1 + \frac{\rho}{2\pi} \int y(X_1, X_3)f(X_1, X_3) \times \left[y(X_3, X_2)f(X_3, X_2) + y(X_3, X_2) - 1 \right] dX_3 \quad (3.4)$$

3.1.2.2 Rose Potential Model

The rose potential is another model first introduced by Williamson et al. (2017). This model, while similar to the three-pronged BN2D, is notably different in that the rose potential model simplifies the model by use of a radial sinusoidal plot to make the three "prongs" of the particle.

3.1.2.3 Two-Dimensional Modeling

Something other than OOPSE? (not seeing obvious answer other than "custom code modified/forked from existing 3D tools")

The Object Oriented Parallel Simulation Engine (OOPSE) was introduced by A. et al. (n.d.) as a relatively light-weight molecular dynamics simulation package focused on "efficiently integrating equations of motion for atom types with orientational degrees of freedom" (from abstract). While OOPSE was further developed and renamed OpenMD, a fork of OOPSE was developed specifically to model water in two dimensions.

3.2 Goal of Project

The objective of this work was to model two-dimensional water with a surface designed to discourage crystal growth at freezing temperatures. The design of the surface was the primary focus. Successfully designing a surface capable of discouraging water ice formation at freezing temperatures would provide valuable information in designing a three-dimensional model of the same type at a reduced computational cost.

Idea: adjust freezing point depression to be freezing point modification.

3.3 Tools and Terms

Either a refresh from intro or a detailed explanation of OOPSE and the reduced terms.

3.3.1 OOPSE in 2D

Detail differences in computation system that Dr. Fennell developed to allow 2D MD(?) on a 3D program.

3.3.2 Reduced Terms: 2D analogues

Detail differences in dimensionality and define the reduced dimensions. Still working on the understanding/equations.

3.4 Designing System

Include: ensemble, thermodynamic variables, box attributes (size, pressure, temp, etc), number of waters, surface (size, spacing between beads of surface, charges, LJ values, etc)

3.4.1 Defining the Surface

Explain how to develop the surface in the program and how to build a custom surface

3.4.1.1 Manipulation of LJ Potential

Manipulate σ and ϵ values to effectively adjust the radius and interaction strength of surface beads.

3.4.1.2 Manipulation of bead spacing

Detail design of optimizing bead spacing for freezing encouragement or disruption.

3.5 Results

Success of freezing point elevation, pending results for freezing point depression

CHAPTER 4

Germanium Compounds and QM Concerns

4.1 Modeling Germanium Compounds

A work in progress - not much research on Germanium exists.

4.1.1 Computational Complexity of Germanium Compounds

Draw-backs of modeling Germanium. Uncommon but still necessary for wetwork.

4.1.2 Literature Review on Relevant Works

Make note of various Germanium modeling research. Make note of tools and methods used.

4.2 The Initial Problem: Germanium Study

During Fall 2017, Dr. Christopher Fennell was approached by Dr. Charles Weinert of OSU to continue a collaborative effort in sampling conformation energies of two germanium-based compounds of interest to Dr. Weinert's work. Seen as an opportunity to train a new graduate student in conformational calculations, this project was delegated to me. The initial focus was to create the two compounds in a 3D modeling program, save a file of each, run a conformation optimization program on a supercomputer, and read the output to report the findings. As detailed below, this work led to impossibilities, curiosities, and inconsistencies that resulted in a general solution and a discovery of a flaw in a popular computational program.

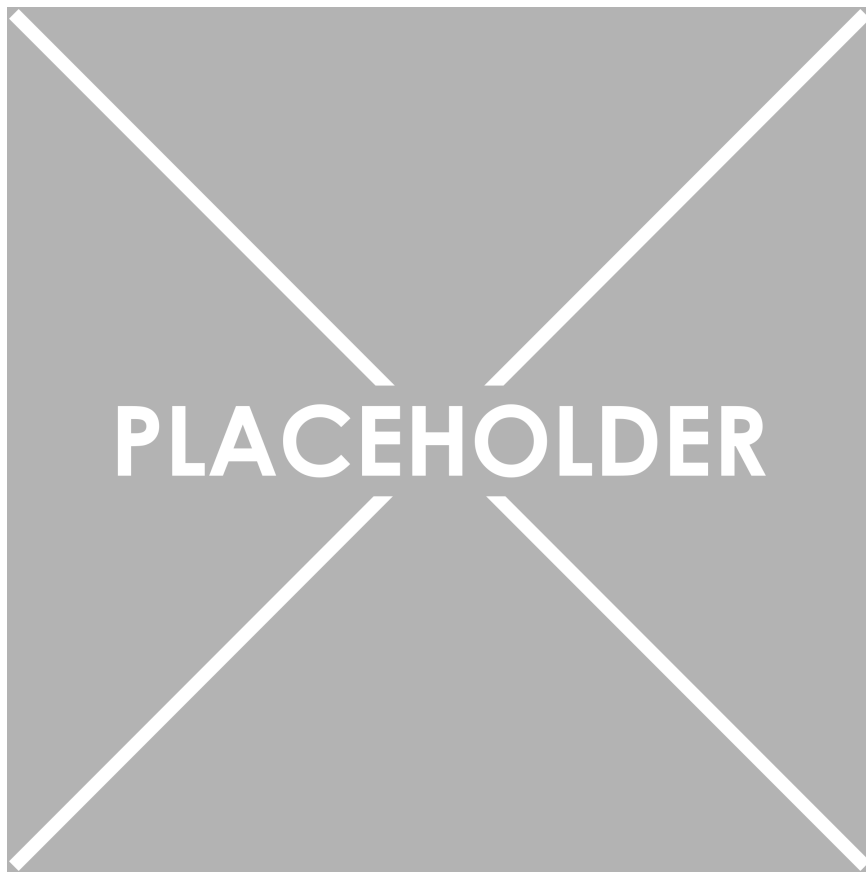


Figure 4.1: Fully trans configuration of pentagermanium-based compound.

4.2.1 Parameters of Work and Previous Collaborator's Results

The two subject germanium-based compounds are very similar: a germanium backbone with terminal isopropyl groups and internal phenyl rings. One compound constituted a pentagermanium chain while the other a hexagermanium backbone. The molecular formula for both is $Pr_3^i Ge(GePh_2)_n GePr_3^i$ where n equals 3 for the pentagermanium or 4 for the hexagermanium compounds, respectively. An example image of both compounds in their fully-trans configurations are provided in figures 4.1 and 4.2.

Dr. Weinert had worked previously with an additional collaborator who provided conformation data supplied in table 4.1. If I want to cite somebody, all I do is type in the citation for Bernal and Fowler (1933).

The approach of labeling the conformation shape of each compound, given the

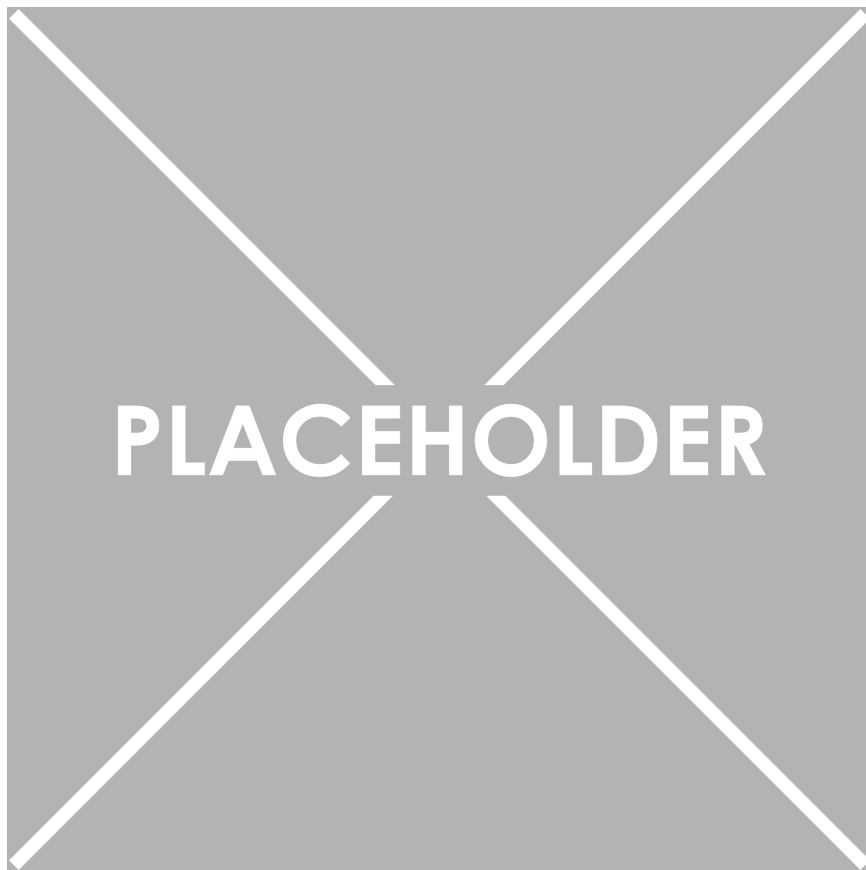


Figure 4.2: Fully trans configuration of hexagermanium-based compound.

Conformation	Energy (E_h)	Δ Energy (E_h)	Δ Energy ($\frac{KJ}{mol}$)
Trans-coplanar	-15014.8403143	0.0066255	17.39525025
Cis-Trans-Cis	-15014.7983311	0.0486087	127.6221418
Trans-Cis-Trans	-15014.8469398	0.0000000	0.0000000
Cis-Trans-Trans	-15014.8246918	0.0222480	58.412124

Table 4.1: Collaborator's Hexagermanium Energies by Conformation
(density functional theory, unknown basis set, energy in Hartrees and KJ/mol)

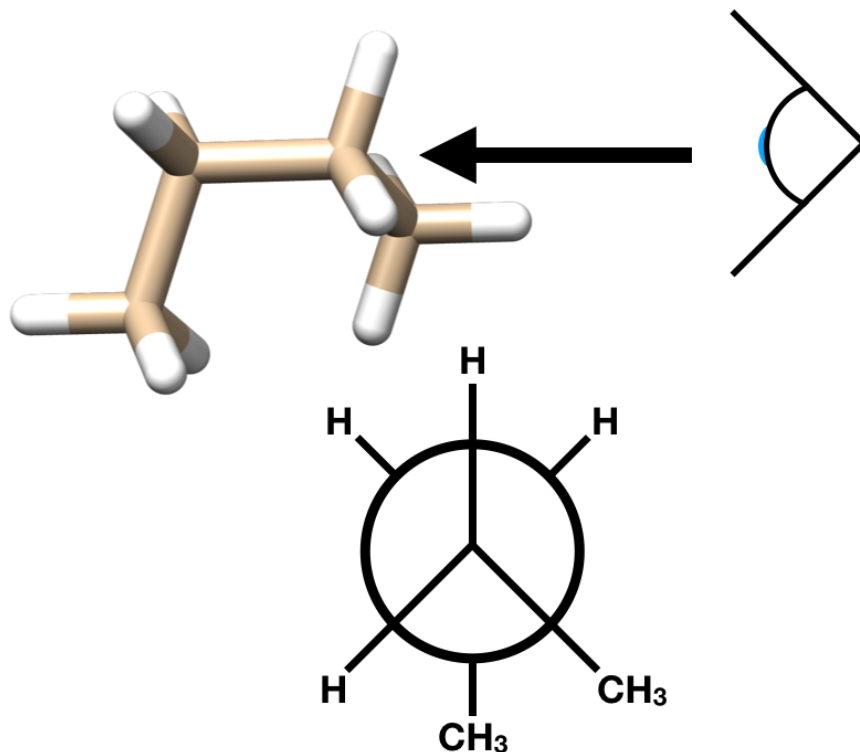


Figure 4.3: Sample Newman projection of cis-butane.

many points of torsion, focuses on the backbone structure. As the raw data from the collaborator was not available, the general dihedral angles of cis and trans proved a vexing focus for initial efforts at conformer design. Using Newman projections like in figure 4.3 as a visual guide, each Ge-Ge bond was defined as cis or trans based on the relative angle produced by the two adjacent bonded Ge atoms to each subject Ge. Specifically, the bonds are marked cis if the most acute angle is 90° or fewer, and likewise trans if greater than 90° up to the maximum 180° . Effectively the cis and trans angles coincide with gauche and anti in organic structure nomenclature. Terminal germanium atoms are not considered as a part of the conformation state. This is partly due to the definition in labeling where the terminal germanium does not have an adjacent germanium for the measured relative angle, in addition to the assumed C_3 symmetry of the terminal Ge with three isopropyl groups reducing the relative effects of terminal germanium rotation. Effectively, only dihedrals formed by four consecutive Ge are given a cis or trans label.

4.2.2 Design and Approach to Solution

The initial approach involved an attempt at basic replication of the collaborative results. As detailed below, the design gradually grew in complexity as a learning process. Eventually, curiosities in results and a desire to automate an objective search algorithm developed into two unique investigations.

4.2.2.1 Design 1: Occam’s Smallest Razer

With each non-terminal Ge-Ge dihedral initially labeled cis or trans for 0° or 180° , about 3 unique pentagermanium and 6 unique hexagermanium structures were built visually on a 3D visualization program (Avogadro). These were rotated without consideration for the phenyl rings populating the non-terminal Ge atoms. Each molecule was subjected to an energy minimization in Gaussian 09 with the B3LYP hybrid function and STO-3G basis set as a single particle in a vacuum at otherwise default settings.

Unsurprisingly, only the fully trans conformers successfully converged (a 22% success rate) into a stable form. These troubles were likely caused by the poor design of the initial conformers. With initial results, the conformer design was altered into a more systematic approach with some consideration for the phenyl rings.

4.2.2.2 Design 2: A Blunt Effort

In the second iteration of the conformer design process, a greater number of backbone conformers were generated. Instead of the simple 180° opposition between the cis and trans conformers, more intentional initial angles seen in Newman projections were selected. Specifically, the anti and both gauche angles were chosen for the natural local minima in a non-bulky molecule, with both gauche angles (60 and 300) labeled as cis and the anti angle (180) as trans. For initial conformer design, these backbone angles were limited to three positions: 60° , 180° , or 300° . For the hexagermanium

compound, these structures were sequentially labeled trans-trans-trans, trans-trans-cis, trans-cis-trans, et cetera until all major unique conformers were produced. For clarity, each conformer was identified by the dihedral angles (60-60-60, 60-60-180) in increasing order (Ge 1-2-3-4, Ge 2-3-4-5, Ge 3-4-5-6 dihedral). The phenyl rings on the non-terminal Ge atoms were left untouched from an initial steepest-descent minimization available from Avogadro ran in the fully trans conformer.

To prevent potentially strong interactions between adjacent phenyl rings, an additional steepest-descent minimization from Avogadro was initially ran with the conformer-defining Ge-Ge dihedral angles locked in place. Additionally, a visual inspection of the phenyl rings and manual adjustments were utilized on Avogadro to reduce the chance of a relatively high energy local minima conformer. The phenyl rings usually were settled in a form of pi stacking or some kind of perpendicular ring interaction, based on relative energy stability according to the immediate simple minimization available.

To further avoid backbone rotation restrictions, variations of the bulky molecules were also produced. These included versions where the phenyl rings were replaced by methyl groups and also where the isopropyl ends were additionally replaced by methyl groups. There intention in these designs were to observe the shift in relative energy between the sets of conformers to determine how significant of a role the phenyl rings and isopropyl groups played. These variations, along with the original form structures, were subject to the same calculations as in the first design: Gaussian 09, B3LYP hybrid functional, STO-3G basis set, no angle restrictions, single particle in a vacuum, otherwise default parameters. The results of these calculations are tabulated in tables 4.2 and 4.3.

Immediately obvious in the table are the considerable number of nonconverged results. An unexpected bulkiness trend followed that a fully methylated variation of the structure was most likely to converge to a stable state, while the fully internal

Internal Species	Terminal Species	Conformer	Final Energy (Hartrees)	Energy (Hartrees)	Energy (KJ/mol)
methyl	methyl	60-60	-10738.91336	0.0000454	0.119
methyl	methyl	60-180	-10738.9134	0	0
methyl	methyl	60-300	-10738.91286	0.0005358	1.407
methyl	methyl	180-60	-10738.91325	0.0001533	0.402
methyl	methyl	180-180	-10738.91335	0.0000475	0.125
methyl	methyl	180-300	-10738.91336	0.0000451	0.118
methyl	methyl	300-60	-10738.91336	0.0000455	0.119
methyl	methyl	300-180	-10738.91287	0.0005357	1.406
methyl	methyl	300-300	-10738.9107	0.002703	7.097
phenyl	methyl	60-60	-11875.15183	0.0001451	0.381
phenyl	methyl	60-180	-11875.15144	0.0005304	1.393
phenyl	methyl	60-300	-11875.15197	0	0
phenyl	methyl	180-60	-11875.14282	0.0091505	24.025
phenyl	methyl	180-180	-11875.15004	0.0019354	5.081
phenyl	methyl	180-300	-11875.15064	0.0013353	3.506
phenyl	methyl	300-60	-11875.06665	0.0853257	224.023
phenyl	methyl	300-180	DNC	DNC	DNC
phenyl	methyl	300-300	-11875.1497	0.0022723	5.966
phenyl	isopropyl	60-60	DNC	DNC	DNC
phenyl	isopropyl	60-180	-12341.23176	0.0053028	13.923
phenyl	isopropyl	60-300	DNC	DNC	DNC
phenyl	isopropyl	180-60	DNC	DNC	DNC
phenyl	isopropyl	180-180	-12341.23513	0.001935	5.08
phenyl	isopropyl	180-300	DNC	DNC	DNC
phenyl	isopropyl	300-60	DNC	DNC	DNC
phenyl	isopropyl	300-180	-12341.23706	0	0
phenyl	isopropyl	300-300	DNC	DNC	DNC

Table 4.2: Data of B2LYP/STO-3G minimization of variations of pentagermane compound at various conformers. DNC denotes a failure to converge with the self-consistent field method.

Internal Species	Terminal Species	Conformer	Final Energy (Hartrees)	Energy (Hartrees)	Energy (KJ/mol)
methyl	methyl	60-60-60	-12870.91834	0.0009503	2.495
methyl	methyl	60-180-60	-12870.91929	0.0000004	0.001
methyl	methyl	60-180-180	-12870.91813	0.0011628	3.053
methyl	methyl	60-180-300	-12870.91869	0.0005972	1.568
methyl	methyl	60-300-300	DNC	DNC	DNC
methyl	methyl	180-60-60	-12870.91897	0.0003189	0.837
methyl	methyl	180-180-60	-12870.91833	0.0009585	2.517
methyl	methyl	180-180-180	-12870.91929	0.0000004	0.001
methyl	methyl	180-180-300	-12870.91929	0.0000003	0.001
methyl	methyl	180-300-60	-12870.91897	0.0003192	0.838
methyl	methyl	300-60-180	DNC	DNC	DNC
methyl	methyl	300-180-60	-12870.91929	0	0
methyl	methyl	300-180-180	DNC	DNC	DNC
methyl	methyl	300-180-300	-12870.91814	0.0011527	3.026
phenyl	methyl	60-60-60	DNC	DNC	DNC
phenyl	methyl	60-60-180	-14385.89674	0.0052183	13.701
phenyl	methyl	60-60-300	-14385.89487	0.0070829	18.596
phenyl	methyl	60-180-60	DNC	DNC	DNC
phenyl	methyl	180-60-60	DNC	DNC	DNC
phenyl	methyl	180-60-180	-14385.90195	0	0
phenyl	methyl	180-60-300	-14385.89855	0.0033998	8.926
phenyl	methyl	180-180-180	-14385.83838	0.0635763	166.92
phenyl	methyl	180-300-180	-14385.79233	0.1096251	287.821
phenyl	methyl	300-60-60	DNC	DNC	DNC
phenyl	methyl	300-60-180	-14385.89836	0.003597	9.444
phenyl	methyl	300-60-300	-14385.89836	0.0035979	9.446
phenyl	methyl	300-180-60	DNC	DNC	DNC
phenyl	methyl	300-300-300	DNC	DNC	DNC
phenyl	isopropyl	60-180-180	-14851.9865	0	0
phenyl	isopropyl	60-300-60	DNC	DNC	DNC
phenyl	isopropyl	60-300-180	DNC	DNC	DNC
phenyl	isopropyl	180-300-60	DNC	DNC	DNC
phenyl	isopropyl	180-300-180	DNC	DNC	DNC
phenyl	isopropyl	180-300-300	DNC	DNC	DNC
phenyl	isopropyl	300-300-60	DNC	DNC	DNC
phenyl	isopropyl	300-300-180	DNC	DNC	DNC
phenyl	isopropyl	300-300-300	DNC	DNC	DNC

Table 4.3: Data of B2LYP/STO-3G minimization of variations of hexagermane compound at various conformers. DNC denotes a failure to converge with the self-consistent field method.

phenyl structures with methyl ends slightly reduced convergence and the original fully internal phenyl structures with isopropyl ends drastically reduced convergence. The common-sense expectation that the addition of the phenyl ends would reduce stability was not parroted in these results. A deeper exploration into the change of stability is a promising avenue for future investigation, but was not further explored in this work. As can be seen in table 4.3, the lowest energy conformer for each structure varied greatly, but never included the fully trans conformer and only once the collaborator-reported trans-cis-trans conformer as the most stable. Still, given the considerable amount of nonconverged conformers, a new design was necessary to further improve the scope of the lowest energy conformation search.

4.2.2.3 Design 3: Death by 1.59 Million Cuts

In the final version of the conformer generation effort, additional creation efforts were focused on the individual phenyl rings. The unfavorable interactions between the phenyl rings were considerable hurdle in the previous designs and a potential explanation for the large number of nonconverged structures, including the possibility that the terminal isopropyl hexagermanium structures contained particularly unfavorable interactions among the phenyl rings. This third design sought to remove the uncertainty in phenyl ring bulkiness by applying the same approach as the backbone generation: create unique conformers of every backbone torsion and phenyl ring, limiting each torsion to one of three rotational positions following the Newman projection style. Unfortunately, this task proved prohibitively large.

As an explanation for the insurmountability of the problem, consider the hexagermanium structure. The germanium dihedrals represent three rotatable bonds each with three initial positions. To include the phenyl rings would require the inclusion of eight new rotatable bonds each with three initial positions. Additionally, considering each terminal germanium’s rotation while ignoring each isopropyl’s rotatable bonds

adds two initial positions each with three initial positions. Together, this creates a structure with 13 rotatable bonds each with three initial positions. The number of conformers follows as $3^{13} = 1,594,323$ initial conformers. Now we must consider the computational aspect of this many conformers. At 10 conformers rotated and generated per second and 16 KB per conformer, the initial conformers would require 44.3 hours and generate 25.49 GB of data just in the initial structures. At an average of 72 minutes per computation and 73.7 MB produced at B3LYP hybrid functional and STO-3G basis set and access to all 255 regular nodes of Oklahoma State University’s Cowboy cluster running in parallel, the complete computation would generate 117.5 TB of data and require 312 days of continuous computation to determine a possible lowest energy conformer of this one molecule at a relatively low level basis set and theory. A request to utilize 100% of university supercomputer resources for nearly a year for the sake of determining the lowest energy conformer of one molecule would likely be rejected, so this task would likely require a time scale of years or even decades to produce with shared access to university resources. While conventionally considered a small molecule, the scale of conformers and computational requirements pushes this problem into the realm of Levinthal’s paradox.

While this third design would have likely revealed the lowest energy conformer, or at least one considerably close the the exactly lowest energy conformer, the effort ultimate fails under its own weight. Even with efforts to truncate duplicate forms, the problem of scale remains. A reduction by 50% still requires a computation effort in the timescale of years or decades for the calculation of a single molecule. For an effective computational outlook, this system needs to be reduced by several orders of magnitude.

4.2.3 Scale Reduction Efforts

For a system with conformers on the millions scale and computations on the hour scale, a magnitude reduction in either aspect would improve the practicality of this design approach. For example, by simplifying the computational method from 72 minutes on average to 5 minutes on average, the overall computational requirement would be reduced by 92%, a full order of magnitude. Unfortunately, reducing the complexity of the method sacrifices the reliability of data. A potential solution here would be to create rounds of calculations at different complexities, where each sequential round restricts the pool of potential conformers. Ideally, the balance of the increasing computational complexity and the decreasing pool size would maintain a consistent computational requirement. For example, a new round using a higher functional theory and basis set at 5x computational requirement would ideally be paired with a reduction in conformer pool size by a factor of 5. This would produce a series of calculation sets with additive computational requirement instead of a magnitudinal expansion.

The natural next question lies within the reliability of basis sets and functional theories. It naturally follows that a less-accurate method should not be relied on while better methods exist. However, considering the scale of the conformer pool, it follows that a less accurate method would still produce energy values with a roughly similar internal consistency. For example, a 180-0-180 form of the hexagermanium compound with parallel phenyl rings as modeled in figure 4.4 will have intense syn interactions between some phenyl rings and will likely not yield a desirable energy value at any level of calculation while a fully trans form with perfect pi stacking phenyl rings will likely have a lower energy value at all levels of calculation. It follows that, at lower levels of accuracy, the extremely high energy conformers can be pruned from the pool early and drastically reduce overall computational requirements. A generic effort at producing a method in this style is detailed in chapter 5, while the remainder of this

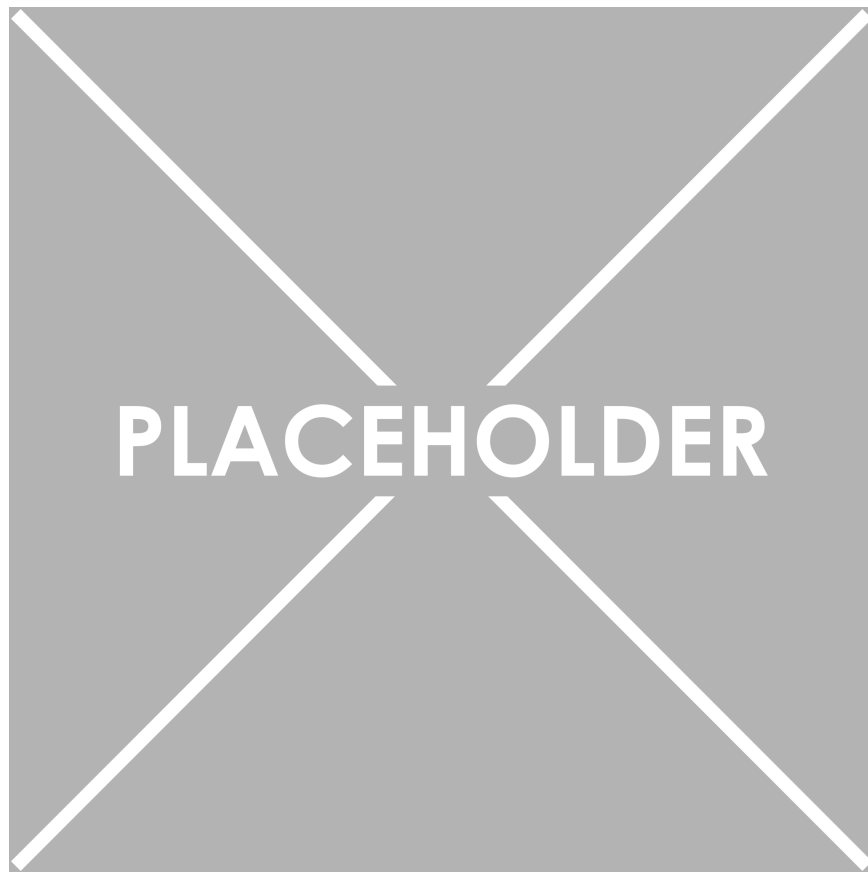


Figure 4.4: Visualization of a trans-cis-trans hexagermane structure.

chapter details additional efforts of calculating these germanium compounds.

4.2.4 Efforts at Simplification

One potential avenue of simplifying the process is computing the energy minimizations of lower-period atoms (e.g. a carbon backbone instead of germanium) and then applying a correction factor for a net reduction in computation time. As a period 4 element, germanium exhibits computational qualities similar to but more complicated than both carbon and silicon. Using tested samples, an energy minimization of a carbon-backbone molecule instead of the germanium represented a 92% reduction in computation speed. Assuming a nominal correction factor exists and can be applied, this represents an order of magnitude reduction in computation time with one simplification. Potentially, this would allow investigators to much more quickly

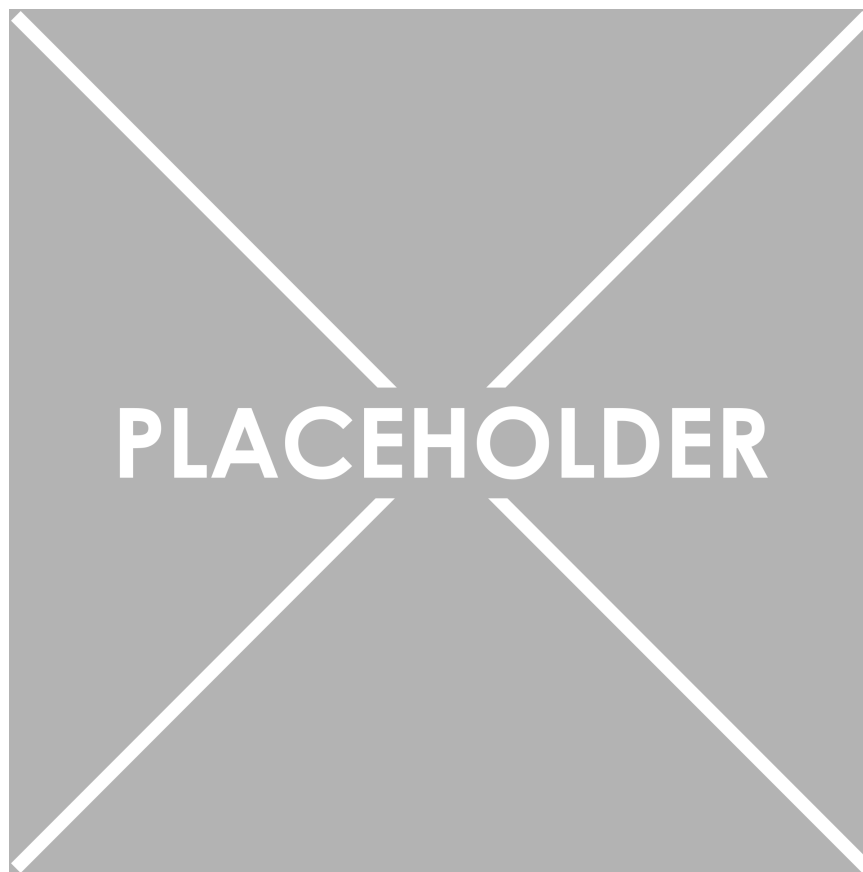


Figure 4.5: Visualization of a multiple pure group IV torsions at various theories and basis sets

eliminate high energy conformers and more rapidly reduce the scope of the search.

The approach to acquiring sufficient data for a possible correction factor involved running an extremely simplified form of the germanium compounds, specifically a butagermanium backbone with hydrogens occupying all terminal and internal bonds. This reduced the complication and complexity of bulkiness and allowed for quick full torsion rotations about the single Ge-Ge-Ge-Ge dihedral. By operating at intervals of 5° , a full torsion drive provides a glimpse at relative energies of the molecule at 72 discrete states. An example plot of this torsion drive is shown in figure 4.5 Once multiple torsion drives had completed in multiple group four elements (isobutyl C, Si, and Ge were all built and tested), the energies could be compared and analyzed for any relative or absolute scaling at the additive or multiplicative reference.

For a full comparative set, 3456 points of analyzed data were generated for each reference molecule’s free energy in comparison with the others. Unsurprisingly, no simple correction factor arose by method of a simple additive or multiplicative term applied toward all torsion points. A visual trend arose with these drives, where curious results arose. A future avenue of research would be to further explore this with depressive or polynomial terms to discover whether a simple corrective function might exist with specific molecules.

While the comparisons between different group 4 elements did not yield positive results, a similar approach with differing functionals and basis sets was also attempted. While this approach likewise did not find any simple correction factor, a graphical representation of multiple functionals across the isobutyl C, Si, and Ge show an interesting trend, as visualized by a graph provided by Dr. Christopher Fennell and shown in figure 4.6

As expected by different types of calculations, the torsion graphs hold different internal relative energies. For carbon, all four functionals produced a clean curve. The AM1 and PM3 functionals produced unexpected results for both Si and Ge graphs. In each, the expected highest energy 0° torsion angle was instead the most favorable of the three eclipsed angles. Additionally, the Si PM3 and the Ge AM1 and PM3 functionals showed strong spikes along the expectedly smooth curve, with the Ge PM3 being noticeably broken. These are likely due to the additional electronic interactions from the additional electron shells of Si and also the d-orbital electrons in Ge. While the Si graphs smoothed out for the B3LYP and HF functionals at STO-3G basis set, the Ge B3LYP showed significant spikes and only the HF STO-3G exhibited a smooth curve. Effectively, this discovery of spikes along torsion drives led to the realization that the validity of a basis set could possibly be determined by the smoothness of a torsion drive. For example, any calculation of a germanium-containing molecule will likely not produce reliable results with a B3LYP hybrid functional and STO-3G basis

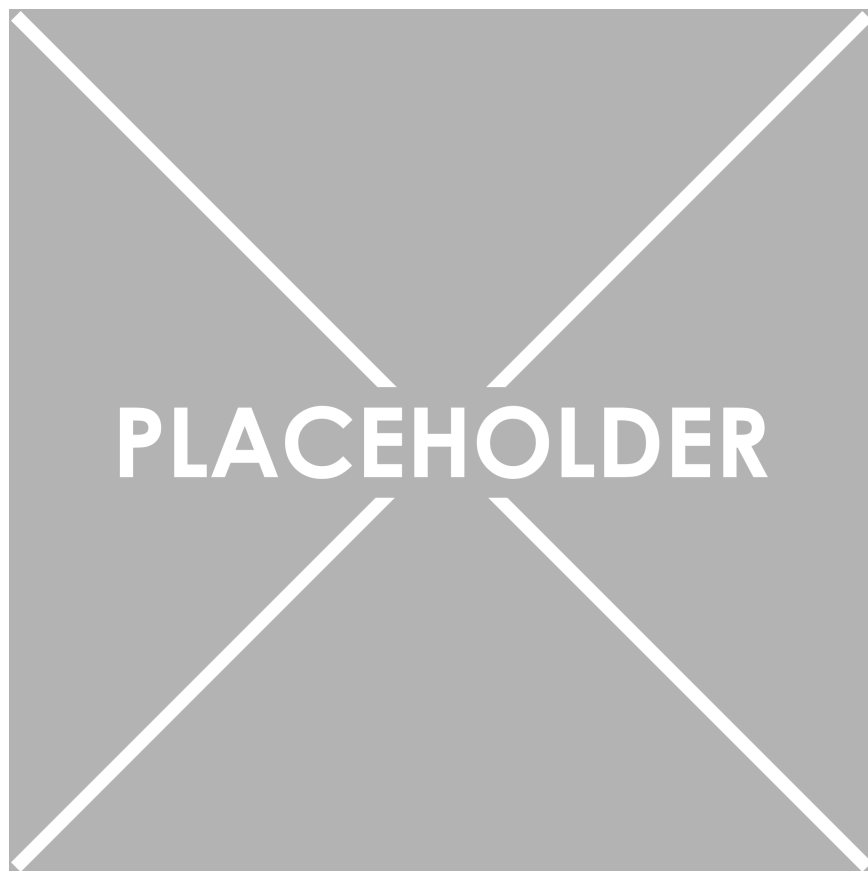


Figure 4.6: Visualization of a multiple pure group IV torsions at various theories and basis sets

set, while the Hartree Fock STO-3G calculation would at least be tentatively reliable for comparative energy levels at various conformations.

The next natural step was to calculate and plot additional functional theories and basis sets with the isobutagermanium chain. While effectively a lightly guided meandering through the available calculation types, the first effort was to observe relative differences across multiple basis sets of the Hartree Fock theory and to examine the relative computational requirements of each. This plan was quickly redirected, however, when a curiosity within the data was revealed.

4.2.5 Discovery of a Consistent Inconsistency

The extended round of torsion drive calculations included an alteration in representation of the data. As the focus had shifted from relative energies and intensities across multiple theories and basis sets to a focus on graph smoothness and internal relative energies, the energy axis of plots were reduced to a unitless scale ranging 0 to 1, where 0 represents the minimum energy and 1 represents the maximum energy in a given set of torsion drive data. This allowed for graphical representations of each torsion drive to emphasize the internal variation of torsions relative to the minimum and maximum values. This was accomplished by taking any set of data with absolute scale energy unit, identifying the minimum and maximum values, and scaling each data point according to equation 4.1. The script to collect and scale data points is detailed in C.

$$E_{i,red} = \frac{E_{i,abs} - E_{min,abs}}{E_{max,abs} - E_{min,abs}} \quad (4.1)$$

Need more: Actually finding the Ge problem, and confirming it.

CHAPTER 5

Sampling Conformation Landscapes by Rotatable Bond Degrees of Freedom

5.1 Introduction to Topic

EACH SUBSECTION: DEFINITION OF TERMS

For ROUGHLY forty years, computational programs have allowed investigators to model chemical systems with high accuracy to determine their physical properties.

5.1.1 A Brief History on Conformation Landscapes

5.1.1.1 Levinthal's Paradox

Discuss history of Levinthal and his paradox. Provide the non-paradoxical solution.

Next: Levinthal golf courses by Ken Dill.

5.1.2 Literature Review on Relevant Works

intro: Reference introduction: inherent complexity of "objective" or "exhaustive" search, levinthal's paradox as it applies to non-biological systems, ELSE?

Run through a set of dihedral positions at a constant interval. Selection of lowest-energy optimization organized on dihedral values. Quick determination of importance of dihedral based on how heavily it impacts internal energy. Splitting "best" dihedral into smaller interval to repeat the process.

This method produces an interesting multilayered visual plot with a zooming effect toward the lowest energy conformer. An example of how this might look for a

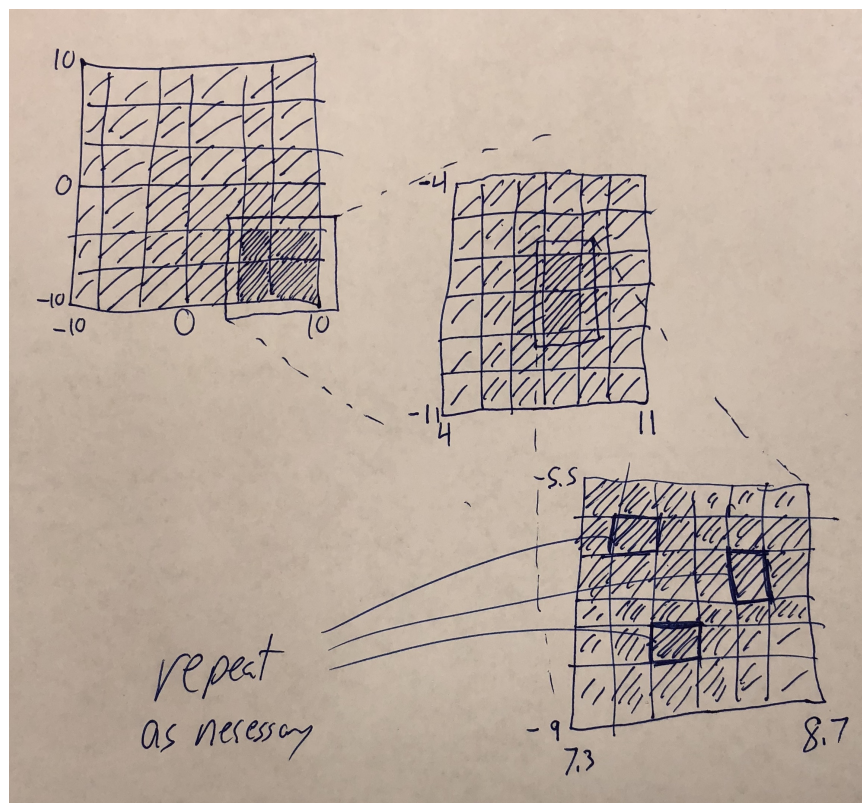


Figure 5.1: example VR chart (hand-drafted, CGI pending)

two-dihedral molecule is given in figure 5.1.

5.2 Design of System

System designed in Python for ease of development and compiled via Cython for computational efficiency. Utilizes Gaussian and UCSF Chimera, but can be redesigned for any computational programs that accomplish the desired tasks. An overview of system flow given in figure 5.2.

This design, with implementation being a current work in progress, but shouldTM work as a cascade toward the lowest energy conformer in each case.

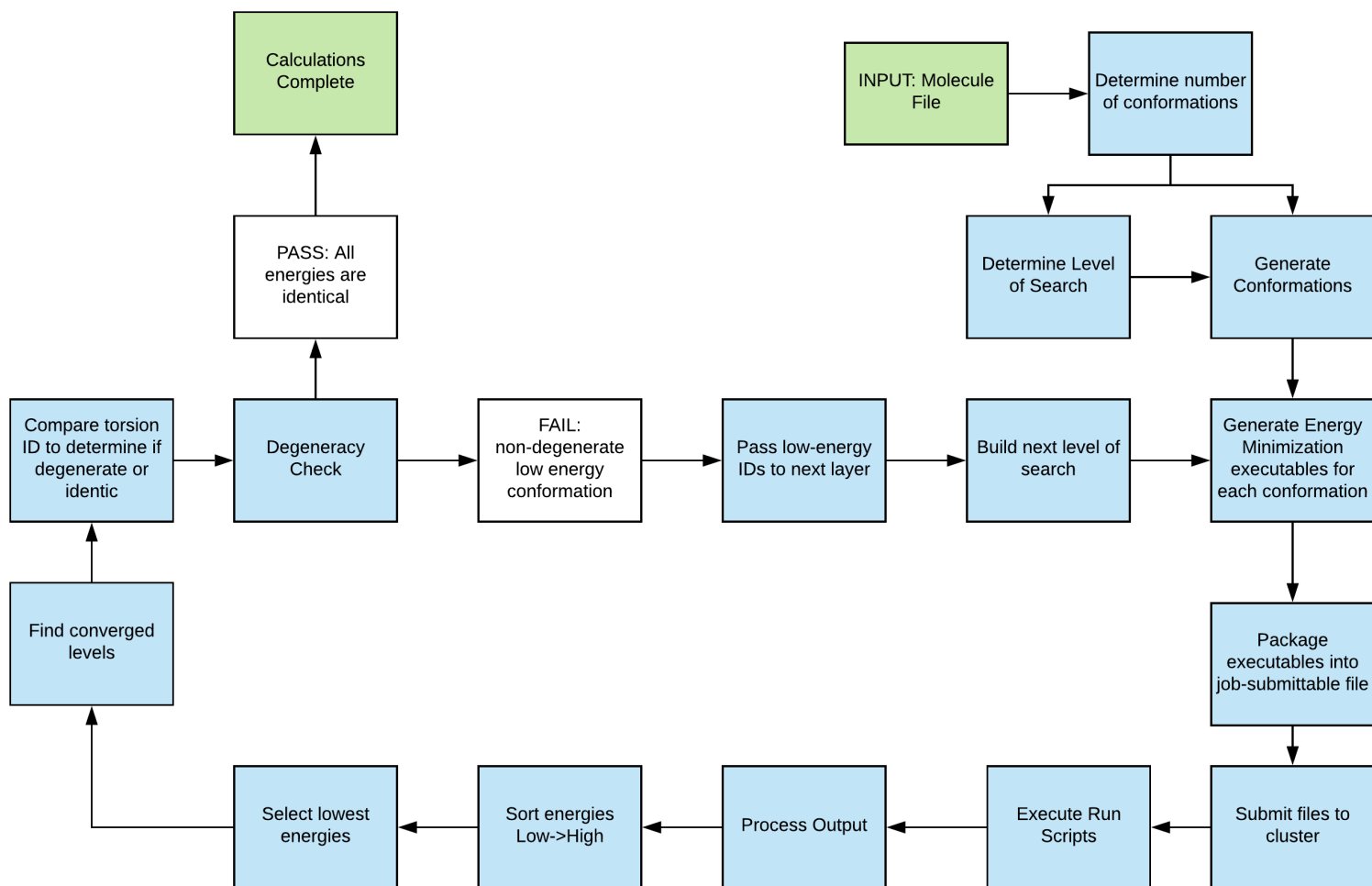


Figure 5.2: Flow of method design for variable resolution conformation landscape search.

5.2.1 Variation of Theory and Basis Set Usage by System Size and largest atom type

System will have inherent restrictions. Give an example of large system with simple atoms, small system with complex atoms. System estimates quantity and cost of calculations based on computational limits defined by user for various theory-basis set pairings. System optimizes calculations for the scale of run (is it the first broad-scope search, or a final near-exact search).

5.2.2 Computational Optimization by Varying Resolution

Extant work not optimized for a general search (negative claim: make sure literature has nothing). Design should work with additional development (primary focus this semester) as a general search tool.

5.2.3 Inherent Complications

Complications of size and atom type, impossible conformers, duplications, limited computational resources.

5.3 Results

Current success: finding accepted lowest energy conformer of a two-dihedral system by manually cranking each step. Self-running is still a work in progress.

5.3.1 Problems

Difficulty in defining an abstract system based on arbitrary hardware limitations. Propose a test-run to determine efficiency and resource availability.

5.3.2 Anticipated Approaches for Future Work

Putting system into a single cohesive program. Further optimizing Theory/Basis Set determination by computational efficiency as well as system size (determine an upper-limit of computation?)

References

A., M. M., F., V. C., Teng, L., J., F. C. and Daniel, G. J. (n.d.), ‘Oops: An object-oriented parallel simulation engine for molecular dynamics’, *Journal of Computational Chemistry* **26**(3), 252–271.

URL: <https://onlinelibrary.wiley.com/doi/abs/10.1002/jcc.20161>

Ben-Naim, A. (1971), ‘Statistical mechanics of ”waterlike” particles in two dimensions. i. physical model and application of the percus-yevick equation’, *The Journal of Chemical Physics* **54**(9), 3682–3695.

URL: <https://doi.org/10.1063/1.1675414>

Bernal, J. D. and Fowler, R. H. (1933), ‘A theory of water and ionic solution, with particular reference to hydrogen and hydroxyl ions’, *The Journal of Chemical Physics* **1**(8), 515–548.

URL: <https://doi.org/10.1063/1.1749327>

Buch, V., Sandler, P. and Sadlej, J. (1998), ‘Simulations of h₂o solid, liquid, and clusters, with an emphasis on ferroelectric ordering transition in hexagonal ice’, *The Journal of Physical Chemistry B* **102**(44), 8641–8653.

URL: <https://doi.org/10.1021/jp980866f>

Williamson, C. H., Hall, J. R. and Fennell, C. J. (2017), ‘Two-dimensional molecular simulations using rose potentials’, *Journal of Molecular Liquids* **228**(Complete), 11–18.

APPENDIX A

Ice Ih to Ice XI Conversion

Listed below is the source code utilized in the conversion of a PDB Ice Ih structure into an Ice XI structure. This code is functional in a Python 2.7 environment with NumPy and SciPy packages included.

A.1 Code: Crystal Disorganizer Tool

```
1
2 #!/usr/bin/python
3
4 # Author = Gentry Smith
5 # Copyright 2016, all rights reserved
6
7 # this reads in a .PDB file , takes an argument for deformities per
   molecules , and randomly organizes the crystal
8 # structure into a disordered proton formation
9
10 # import sample: python PDBDisorganize.py arg1 arg2 arg3
11 # where:
12 # arg1 = source pdb file to be read (ex: acetone.pdb or acetone)
13 # arg2 = number of defects per molecule (in H2O, num of non-hydrogen-
   bonds. from 0 to 4)
14 # arg3 = desired output pdb file name
15
16 import sys
17 print sys.path
18 import string
19 import numpy as np
20 import math
21 import random
22
23 sys.setrecursionlimit(10000000) # maximum recursive depth. Set to
   (10,000,000) as under maximum
24
25
26 pdbIN = file(sys.argv[1])      # source PDB file
27 maxErr = int(sys.argv[2])      # max errors allowed
28 pdbOUT = str(sys.argv[3])      # output file name
29 finalData = [ [ [ 0 for i in range(3) ] for j in range(3) ] for k in
   range(300) ]
30
31 # looks at args validity
32 def checkArgs(arg1, arg2, arg3):
33     returnBool = False
```

```

34     if type(arg1) != file: # check arg1
35         print"Bad arg", arg1, " must be a file "
36         returnBool = True
37     if type(arg3) != str: # check arg3
38         print"Bad arg", arg3, ", must be a file name"
39         checkPDBSuffix(arg3)
40         print arg3
41         returnBool = True
42     if type(arg2) != int: # check arg2 type
43         print "Bad arg2: ", arg2, " is not an int."
44         returnBool = True
45     elif type(arg2) == int:
46         if arg2 < 0 or arg2 > 4: # check arg2 range
47             print "arg2 is not in a valid range 0 <= arg2 <= 4"
48             returnBool = True
49     return returnBool
50
51 def checkPDBSuffix(pdbFile):
52     if string.find(pdbFile, '.pdb', 0, len(pdbFile)) == -1:
53         print("did not find 'pdb' in ", pdbFile, ". Appending...")
54         pdbFile += '.pdb'
55
56
57
58 # reads in file ,
59 def readFile(fileName):
60     print "Reading file..."
61     # gets number of atoms
62     atoms = 0
63     for line in fileName:
64         data = line.split()
65         if len(data) > 0:
66             if data[0] != "CONECT" and data[0] != "END":
67                 atoms += 1
68     # print "atoms: ", atoms
69     numMol = atoms / 3 # assumes 3-atom water molecule
70     dataTable = [ [ [ 0 for i in range(3) ] for j in range(3) ] for k in
71                     range(numMol) ]
72     fileName.seek(0)
73     iter0 = 0
74     iter1 = 0
75     pdbType = -1
76     for line in fileName:
77         data = line.split()
78         if pdbType == -1:
79             if data[0] == "ATOM":
80                 pdbType = 0
81             elif data[0] == "HETATM":
82                 pdbType = 1
83         # print "LineTuple=", data
84         if len(data) > 1 and ( data[0] == "ATOM" or data[0] == "HETATM"
85                                ):
86             if data[0] == "ATOM":
87                 newData = getDataATOM(data)

```

```

86         for i in range(3):
87             #data[molecule][atom][X/Y/Z]
88             dataTable[iter0][iter1 % 3][i] = newData[i]
89         elif data[0] == "HETATM":
90             dataTable[iter0][iter1 % 3] = getDataHETATM(data)
91         if iter1 == 2:
92             iter0 += 1
93             iter1 = 0
94         elif iter1 != 2:
95             iter1 += 1
96     # print "DataTable: ", dataTable
97     print "File read"
98     return dataTable, pdbType
99
100
101     # Split by index
102     # if having a problem with reading data, check .pdb to see if data
103     # has a space between each value
104 # reads XYZ coordinate data from ATOM-type pdb
105 def getDataATOM(strLine):
106     # print "Getting ATOM Data..."
107     dataLine = strLine[5:8]
108     # print "dataline: ", dataLine
109     i = 0
110     while i < 3:
111         # print "dataline[", i, "]: ", dataLine[i]
112         dataLine[i] = float(dataLine[i])
113         # print "dataline[", i, "]" type: ", type(dataLine[i])
114         i += 1
115     return dataLine
116
117
118 # reads XYZ coordinate data from HETATM-type pdb
119 def getDataHETATM(strLine):
120     # print "Getting HETATM Data..."
121     dataLine = strLine[5:8]
122     # print "dataline: ", dataLine
123     i = 0
124     while i < 3:
125         # print "dataline[", i, "]: ", dataLine[i]
126         dataLine[i] = float(dataLine[i])
127         # print "dataline[", i, "]" type: ", type(dataLine[i])
128         i += 1
129     return dataLine
130
131
132 # gets all four position vectors of hydrogen/lone pair as offset of
133 # oxygen molecule
134 def getOrientations( molecule ):
135     # 120 degrees = ( 2 * pi ) / 3 radians
136     theta = ( ( 2 * math.pi ) / 3 )
137     newMol = zeroOrientation(molecule)
138     returnInt1 = rotateMolecule(newMol[1], newMol[2], theta)

```

```

138     returnInt2 = rotateMolecule(newMol[1], newMol[2], (-1 * theta) )
139     return [returnInt1, returnInt2]
140
141
142 # randomly selects new orientation, returns two unique ints, from 0 to 3
    inclusively
143 def newRandOrientation( positions ):
144     # print "Changing orientation"
145     randVal1 = random.randint(0,3)
146     randVal2 = random.randint(0,3)
147     while randVal1 == randVal2:
148         randVal2 = random.randint(0,3)
149     newMol = [ [ 0, 0, 0 ],
150                positions[ randVal1 ],
151                positions[ randVal2 ] ]
152     return newMol
153
154 # selects new orientation from list. Reduces computational overhead in
    re-orientation option traversal
155 def newSetOrientation( positions, pos1, pos2 ):
156     newMol = [ [ 0, 0, 0 ],
157                positions[ pos1 ],
158                positions[ pos2 ] ]
159     return newMol
160
161
162 # sets molecule coordinates so that oxygen is the origin
163 def zeroOrientation(source):
164     # print "Zeroing Molecule..."
165
166     oxy = source[0]
167     hyd1 = source[1]
168     hyd2 = source[2]
169
170     # print "Oxygen pos: ", oxy
171     # print "Hydrogen 1: ", hyd1
172     # print "Hydrogen 2: ", hyd2
173
174     zeroedOrigin = [0, 0, 0]
175     zeroedHyd1 = [0, 0, 0]
176     zeroedHyd2 = [0, 0, 0]
177     for i in range(3):
178         zeroedHyd1[i] = hyd1[i] - oxy[i]
179         zeroedHyd2[i] = hyd2[i] - oxy[i]
180
181     # print "Zeroed Hydrogen 1: ", zeroedHyd1
182     # print "Zeroed Hydrogen 2: ", zeroedHyd2
183
184     # return new molecule position
185     newMol = [zeroedOrigin, zeroedHyd1, zeroedHyd2]
186     return newMol
187
188 # resets the zeroed molecule to the original oxygen position
189 def resetOrientation(oxygenPos, molecule):

```

```

190 # print "Resetting molecule..."
191 rO = oxygenPos
192 rH1 = [0,0,0]
193 rH2 = [0,0,0]
194 newMol = []
195 for i in range(3):
196     rH1[i] = molecule[1][i] + rO[i]
197     rH2[i] = molecule[2][i] + rO[i]
198     newMol = [rO, rH1, rH2]
199 # print "Rebuilt Molecule: ", newMol
200 return newMol
201
202 # rotates vector about axis for theta degrees
203 # Handler for rotationMatrix function below
204 def rotateMolecule(vector, axis, theta):
205     rotMatx = rotationMatrix(axis, theta)
206     return np.dot(rotMatx, vector)
207
208
209 # Creates Rotation matrix for a given axis and theta
210 # from stackoverflow user unutbu
211 # page: http://stackoverflow.com/questions/6802577/python-rotation-of-3d-vector
212 def rotationMatrix(axis, theta):
213     """
214
215     :type axis: list
216     :type theta: union
217     """
218     axis = np.asarray(axis)
219     theta = np.asarray(theta)
220     axis /= math.sqrt(np.dot(axis, axis))
221     a = math.cos(theta/2.0)
222     b, c, d = -axis*math.sin(theta/2.0)
223     aa, bb, cc, dd = (a * a), (b * b), (c * c), (d * d)
224     bc, ad, ac, ab, bd, cd = (b * c), (a * d), (a * c), (a * b), (b * d), (c * d)
225     return np.array([ [ (aa + bb - cc - dd), ( 2 * ( bc + ad ) ), ( 2 *
226         ( bd - ac ) ) ],
227         [ ( 2 * ( bc - ad ) ), (aa + cc - bb - dd), ( 2 *
228         ( cd + ab ) ) ],
229         [ ( 2 * ( bd + ac ) ), ( 2 * ( cd - ab ) ), (aa +
230         dd - bb - cc) ] ] )
231
232 # gets results from rotateAboutAxis plus two Hydrogens to get the
233 # tetrahedron positions
234 def getTetrahedronPositions(molecule):
235     positions = [ [ 0 for i in range(3) ] for j in range(4) ]
236     newMol = zeroOrientation(molecule) # zero molecule
237     positions[0] = newMol[1]
238     positions[1] = newMol[2]
239     newPos = getOrientations(molecule) # get final two positions
240     positions[2] = list(newPos[0])

```

```

238     positions[3] = list(newPos[1])
239     return positions                                # return all four positions
240
241
242 # checks distance of new positions from zero
243 def checkDist(posArray):
244     distance = [0 for i in range(len(posArray))]
245     for i in range(len(posArray)):
246         distance[i] = ( (posArray[i][0] * posArray[i][0]) +
247                        (posArray[i][1] * posArray[i][1]) +
248                        (posArray[i][2] * posArray[i][2]) )
249         # print "Distance", i, ": ", distance[i]
250     avg = 0
251     for i in range(len(posArray)):
252         avg += distance[i]
253     averageDistance = ( avg / len(posArray) )
254     # print "Average Distance: ", averageDistance
255     return averageDistance
256
257
258 # prints data given a 3D table of water molecules
259 def printData(data):
260     print "Data: "
261     strData = [" O", "H1", "H2"]
262     dimData = ["X", "Y", "Z"]
263     bigAvg = 0
264     numAtoms = 0
265     for mol in range(len(data)):
266         for atom in range(len(data[mol])):
267             printStr = str(mol) + ": " + strData[atom] + ": "
268             for dimension in range(3):
269                 printStr += dimData[atom] + ":" + "{:7.3f}".format(data[
270 mol][atom][dimension]) + "\t"
271             print printStr
272             bigAvg += checkDist(zeroOrientation(data[mol])[1:])
273             numAtoms += 1
274             print ""
275     print "total average distance: ", bigAvg / numAtoms
276
277 # checks validity of molecule
278 def isDefectiveCheck(err, neighborData, posData, index):
279     # find nearby molecules (avg oxygen distance???)
280     print "checking for defects at index", index, "..."
281     print "neighbor indices: ", neighborData[index]
282     returnBool = False
283     neighbors = 4
284     for i in range(4): # count real neighbors
285         if neighborData[index][1][i] == -1:
286             neighbors -= 1
287     if neighbors <= err: # de facto good if num(neighbors) <
288 maxErrAllowed
289     # print "Fewer neighbors than allowed errors. de facto Good
290 Orientation"

```

```

289     returnBool = True
290     elif neighbors > err: # enough neighbors to require check
291         # print "More neighbors than error threshold"
292         defectCount = 0
293         for neighbor in range(4): # check each neighbor
294             if neighborData[index][1][neighbor] != -1: # skip over non-
existant neighbors
295                 molA = posData[index]
296                 molB = posData[ neighborData[index][1][neighbor] ]
297                 oxyDist = getDistBetweenAtoms(molA[0], molB[0])
298
299                 if minHydrogenDistance(molA, molB) > oxyDist: # check
for facing lone pairs
300                     print "Double Lone Pair defect"
301                     defectCount += 1
302                     break
303                 else: # check for facing protons
304                     smallerHydrogenDistanceCount = 0
305                     isDefective = False
306                     for first in range(2):
307                         if not isDefective:
308                             for second in range(2):
309                                 newDist = getDistBetweenAtoms(molA[ first
+ 1], molB[second + 1])
310                                 if newDist < oxyDist:
311                                     smallerHydrogenDistanceCount += 1
312                                 if smallerHydrogenDistanceCount > 1:
313                                     print "Double Hydrogen defect"
314                                     defectCount += 1
315                                     isDefective = True
316                     # print "Defects found:", defectCount
317                     if defectCount > 4:
318                         print "IMPOSSIBLE AMOUNT OF DEFECTS DETECTED!!!!"
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
319                     if defectCount > err:
320                         # print "Found a bad molecule!"
321                         returnBool = False
322                     else:
323                         # print "Molecule is within parameters."
324                         returnBool = True
325
326     return returnBool
327
328
329 # randomly re-reorients molecule and neighbors, rechecks all
330 def rerunMolAndNeighbors(err, neighborData, posData, index):
331     # print "Re-reordering molecule at", index
332     # err - max errors allowed
333     # neighborData - int[4] of neighbor indices
334     # posData - array of all molecule position vectors
335     # index - location of focus molecule in posData
336     isGood = False
337     timeCount = 0
338     while not isGood:

```



```

339         # re-rotate molecule through all positions (iterated through all
orientations)
340         positions = getTetrahedronPositions(posData[index])
341         zeroedMol = newRandOrientation(positions)
342         # print "isGood CHECK", isGood
343         isGood, posData = iterThroughRotations(err, neighborData,
posData, index)
344         posData[index] = resetOrientation(posData[index][0], zeroedMol)
345         if timeCount >= 13: # { (1 - 1/6)^n < 0.05 } says n = 17
346             # BROKEN - need to rebuild
347             # 0. evaluated molecule has too many defects
348             # 1. reorient molecule statistically probable amount of
times to cover all orientations
349             # 2. Repeat 1. with neighbor 1
350             # 2a repeat 1. with original molecule
351             # 3. Repeat 2. with neighbor 2, 3, 4, as/if necessary
352             for neighborIndex in range(4):
353                 if neighborData[index][1][neighborIndex] != -1:
354                     positions = getTetrahedronPositions(posData[
neighborIndex])
355                     zeroedMol = newRandOrientation(positions)
356                     posData[neighborIndex] = resetOrientation(posData[
neighborIndex][0], zeroedMol)
357                     # isGood = isDefectiveCheck(err, neighborData,
posData, neighborIndex)
358                     isGood = isDefectiveCheck(err, neighborData, posData, index)
359                     if not isGood:
360                         isGood, posData = rerunMolAndNeighbors(err, neighborData
, posData, neighborData[index][1][neighborIndex])
361                     finalData = posData
362                     return True, finalData
363
364 # iterates molecule through all possible rotations
365 def iterThroughRotations(err, neighborData, posData, index):
366     isGood = False
367     pos1 = 0 # tetrahedral position for H1
368     pos2 = 0 # tetrahedral position for H2
369     while not isGood or (pos1 != 3 and pos2 != 3): # iterates through
all orientations, stops if good orientation
370         if pos1 != pos2:
371             posData[index] = newSetOrientation(posData[index][0], pos1,
pos2)
372             isGood = isDefectiveCheck(err, neighborData, posData, index)
373             if pos2 < 3:
374                 pos2 += 1
375             elif pos2 == 3:
376                 if pos1 < 3:
377                     pos1 += 1
378                 pos2 = 0
379             return isGood, posData
380 # determines minimum hydrogen distance between two atoms
381 def minHydrogenDistance(mol1, mol2):
382     minDist = 100
383     for first in range(2):

```

```

384         for second in range(2):
385             newDist = getDistBetweenAtoms(mol1[first+1], mol2[second+1])
386             if newDist < minDist:
387                 minDist = newDist
388         return minDist
389
390
391
392
393
394 # finds neighboring molecules of each molecule
395 def getNeighbors(data):
396     returnData = [ [ [ 0 for i in range(4) ] for j in range(2) ] for k
397                     in range(len(data)) ] # data[molecule][distance,index][four values]
398     for mol1 in range(len(data)):
399         minDist = [100, 100, 100, 100]
400         minIndex = [0, 0, 0, 0]
401         for mol2 in range(len(data)):
402             if mol1 != mol2:
403                 newMin = getDistBetweenAtoms(data[mol1][0], data[mol2
404                                                 ][0])
405
406                 bigIndex = indexOfBiggest(minDist)
407                 if newMin < minDist[bigIndex]:
408                     minDist[bigIndex] = newMin
409                     minIndex[bigIndex] = mol2
410
411                 for i in range(4):
412                     if minDist[i] >= 9:
413                         minDist[i] = -1
414                         minIndex[i] = -1
415
416                 # print "Four smallest Distances of", mol1, ": ", minDist
417                 # print "Four smallest Indices of", mol1, ": ", minIndex
418                 returnData[mol1] = [minDist, minIndex]
419         return returnData
420
421
422
423
424
425 # finds distance between oxygen atoms
426 def getDistBetweenAtoms( mol1, mol2 ):
427     distance = ( ( ( mol1[0] - mol2[0] ) * ( mol1[0] - mol2[0] ) ) +
428                 ( ( mol1[1] - mol2[1] ) * ( mol1[1] - mol2[1] ) ) +
429                 ( ( mol1[2] - mol2[2] ) * ( mol1[2] - mol2[2] ) ) )
430     return distance
431
432
433
434 # gets index of largest item from a list
435 def indexOfBiggest(check):
436     bigIndex = 0
437     for i in range(len(check)):
438         if check[i] > check[bigIndex]:
439             bigIndex = i
440     return bigIndex
441
442
443
444 # writes data to PDB file
445 def writeDataPDB(data, pdbType):

```

```

436     print "Writing Data to", str(pdbOUT)
437     fileName = str(pdbOUT)
438     output = open(fileName, 'w')
439     if pdbType == 0:
440         writeDataPDBATOM(data, output)
441     elif pdbType == 1:
442         writeDataPDBHETATM(data, output)
443     output.close()
444
445
446 # Writes data to PDB file style = ATOM
447 def writeDataPDBATOM(data, inFile):
448     iterator = 0
449     for molecule in range(len(data)):
450         for atom in range(3):
451             iterator += 1
452             outStr = "ATOM "
453             outStr += str(iterator)
454             while len(outStr) < 11:
455                 outStr = outStr[:6] + " " + outStr[6:]
456             outStr += " "
457             if atom == 0:
458                 outStr += " O " + " WAT"
459             elif atom == 1:
460                 outStr += " H1 " + " WAT"
461             elif atom == 2:
462                 outStr += " H2 " + " WAT"
463             outStr += str(molecule)
464             while len(outStr) < 26:
465                 outStr = outStr[:20] + " " + outStr[20:]
466             outStr += " "
467             outStr += "{:8.3f}".format(data[molecule][atom][0])
468             outStr += "{:8.3f}".format(data[molecule][atom][1])
469             outStr += "{:8.3f}".format(data[molecule][atom][2])
470             outStr += " 1.00" + " 0.00"
471             outStr += " "
472             if atom == 0:
473                 outStr += " O "
474             elif atom == 1:
475                 outStr += " H "
476             elif atom == 2:
477                 outStr += " H "
478             outStr += "\n"
479             inFile.write(outStr)
480
481
482 # Writes data to PDB file style = HETATOM
483 def writeDataPDBHETATM(data, inFile):
484     iterator = 0
485     for molecule in range(len(data)):
486         for atom in range(3):
487             iterator += 1
488             outStr = "HETATM"
489             outStr += str(iterator)

```

```

490         while len(outStr) < 11:
491             outStr = outStr[:6] + " " + outStr[6:]
492         outStr += " "
493         if atom == 0:
494             outStr += " O " + " WAT"
495         elif atom == 1:
496             outStr += " H1 " + " WAT"
497         elif atom == 2:
498             outStr += " H2 " + " WAT"
499         outStr += str(molecule)
500         while len(outStr) < 26:
501             outStr = outStr[:20] + " " + outStr[20:]
502         outStr += " "
503         outStr += "{:8.3f}".format(data[molecule][atom][0])
504         outStr += "{:8.3f}".format(data[molecule][atom][1])
505         outStr += "{:8.3f}".format(data[molecule][atom][2])
506         outStr += " 1.00" + " 0.00"
507         outStr += " "
508         if atom == 0:
509             outStr += " O "
510         elif atom == 1:
511             outStr += " H "
512         elif atom == 2:
513             outStr += " H "
514         outStr += "\n"
515         inFile.write(outStr)
516
517
518 # runs program
519 def testRun(inFile, err, outFile):
520     print "Running Test Version of Program..."
521
522
523 # this is the parent runner for the program
524 def runPgm(inFile, err):
525     print "Running Program..."
526     data, pdbType = readFile(inFile)
527     newData = [ [ [ 0 for i in range(3) ] for j in range(3) ] for k in
528                 range(len(data)) ]
529     print "Reordering Molecules..."
530     for i in range(len(data)):
531         positions = getTetrahedronPositions(data[i])
532         zeroedMol = newRandOrientation(positions)
533         newMol = resetOrientation( data[i][0], zeroedMol )
534         newData[i] = newMol
535     print "Molecules Reordered"
536     connectedMolecules = getNeighbors(newData) # -1 index = not
537     neighboring
538     finalData = newData
539     for i in range(len(connectedMolecules)):
540         # print "check defects"
541         isFine = isDefectiveCheck(err, connectedMolecules, finalData, i)
542         # print "isFINE CHECK", isFine
543         if not isFine:

```

```

542         # print "fixing defects"
543         while not isFine:
544             # print "RerunMol"
545             isFine, finalData = rerunMolAndNeighbors(err,
connectedMolecules, finalData, i)
546             # print "rerunDone"
547             writeDataPDB(finalData, pdbType)
548             # printData(newData)
549
550
551 badArgs = checkArgs(pdbIN, maxErr, pdbOUT) # stop in case of bad
argument
552
553 # check input args
554 if not badArgs: # stop in case of bad argument
555     print "Good Arguments, Initializing Reorientation with", maxErr, "
maximum defects"
556     # testRun(pdbIN, maxErr, pdbOUT)
557     runPgm(pdbIN, maxErr)
558 elif badArgs:
559     print "Bad Arguments, Quitting..."

```

APPENDIX B

Two-Dimensional Rose-Potential Water

Below is the python script used to adjust the Rose-Potential system for various interfaces.

B.1 Code: Surface Adjustment Tool

This tool was created to adjust surface information in a given system.

```
1 #!/usr/bin/python
2
3 #### Author: Gentry Smith, Oklahoma State University
4 #### Created: March 11, 2018 4AM
5 #### Last Edited: March 11, 2018
6
7 #### Edits a prepared oopse rose water simulation and creates the desired
8 surface by
9 #### generating a new water.md file labeled newWater.md.
10 #### This adjusts distance and quantity for one or two atom types.
11
12 # input:
13 # oopseSurfaceMaker.py [quantity of atoms] [distance between atoms] [
14 quantity of atom types]
15
16 import sys
17
18 def NoArgRunner():
19     print('Easy input:\npython oopseSurfaceMaker.py [quantity of atoms]
20 [distance between atoms] [quantity of atom types] ')
21     # print('Welcome to the OOPSE rose surface generator! This adjusts
22 the surface to fit your desired system.')
23     print("\nThis script allows up to two types of atoms on the surface
24 .\n")
25     # print('Entering non-int values will quit the program.\n')
26     # try:
27     #     numAtoms = int(raw_input("input the int value of the number of
28 atoms:\n"))
29     #     numTypes = int(raw_input("input the int value of the number of
30 atom types:\n"))
31     # except ValueError:
32     #     print("\n\n\nNot an int. Quitting. . . \n\n\n")
33     #     quit()
34     # for i in range(1, numTypes + 1):
35     #     raw_input('Looking at atom type ' + str(i) + '.\n ')
```

```
def ExtractData(data):
```

```

31     inFile = open(data, 'r')
32     dataCheck = 0
33     afterData = []
34     # print('Extracting Data...')
35     for line in inFile:
36 # print(" " + line + " ")
37 # print(line[:9])
38         # print('line=' + str(line))
39         # print('line.split()=' + str(line.split()))
40         # print('line.split()[1]=' + str(line.split()[1]))
41         if (str(line[:9]) == 'molecule{') and (dataCheck < 2):
42             #print('FOUND: ' + line)
43             dataCheck += 1
44             elif dataCheck >= 2:
45 #                 print(line)
46             afterData.append(str(line))
47
48 # print(str(inTorsions))
49 # print('Done.')
50     return afterData
51
52
53 def GetPosData(data):
54     numAtoms = data[0]
55     distAtoms = data[1]
56     centerIndex = -1
57     atomPosData = [0]
58     dist = 0.0
59     for i in range(1,numAtoms):
60         dist = dist + distAtoms
61         atomPosData.append(float(dist))
62     halfDist = dist / 2
63     for i in range(len(atomPosData)):
64         atomPosData[i] = (atomPosData[i] - halfDist)
65     return atomPosData
66
67     ### creates this data string for each atom:
68     # atom[i]{
69     #     type = "X";
70     #     position( P, 0.0, 0.0 );
71     # }
72 def GetPosDataString(PosData, numTypes):
73
74     # init vars
75     posString = []
76     typeIter = 0
77
78     # make type strings
79     t1 = "HEAVY0"
80     t2 = "HEAVY1"
81     # t3 = "HEAVY2"
82     atomTypes = [ t1, t2 ]
83
84     for i in range(len(PosData)):

```

```

85         atomString = '    atom[' + str(i) + ']{\n        type = "' + \
86                     + str(atomTypes[typeIter]) + '";\n        position( ' + \
87                     + str(PosData[i]) + ', 0.0, 0.0 );\n    }'
88     if typeIter == (numTypes - 1):
89         typeIter = 0
90     else:
91         typeIter = typeIter + 1
92     posString.append(atomString)
93
94     # make trailing data
95     iterStr = '0'
96     for i in range(1, len(PosData)):
97         iterStr = (str(iterStr) + ', ' + str(i))
98     posString.append("\n    rigidBody[0]{\n        members(" + str(iterStr) +
99     ");\n    }\n")
100     posString.append("}\n\nmolecule{\n")
101
102     return posString
103
104
105 def WriteWaterMDDData(PosString):
106     # define standard string data
107     headString = '#ifndef _WATER_MD_\n#define _WATER_MD_\n\nmolecule{\n
108     name = "PART_WALL";\n\n'
109     tailString = ExtractData('water.md')
110     newWater = open('newWater.md', 'w')
111     finalWrite = [headString]
112     for each in PosString:
113         finalWrite.append(each + '\n')
114     for line in tailString:
115         finalWrite.append(line)
116     for i in finalWrite:
117         #if i[-2:] != '\n':
118             # i = i + '\n'
119         newWater.write(i)
120     newWater.close()
121
122
123 def ArgRunner():
124     # ingest input data
125     try:
126         numAtoms = int(sys.argv[1])
127         distAtoms = float(sys.argv[2])
128         numTypes = int(sys.argv[3])
129         # atomTypeData = []
130
131         # for i in range(1, numTypes + 1):
132         #     newData = [ int( sys.argv[ (3+(i*1)) ] ), int( sys.argv[
133         (3+(i*2)) ] ) ]
134         #     atomTypeData.append(newData)
135     except IndexError:
136         print('\n\nImproper input format. Quitting...\n\n')

```



```

136         exit()
137
138     # Figure out surface atom positions and write new water.md file
139     atomPositions = GetPosData([ numAtoms, distAtoms, numTypes ])
140     posString = GetPosDataString(atomPositions, numTypes)
141     WriteWaterMDDData(posString)
142
143     for i in range(numTypes):
144         print('To change the HEAVY' + str(i) + ' charge, enter:\n')
145         print("sed 's\\// HEAVY0" + str(i) + "          0.0\\HEAVY0
146         X.X\\g' DUFF2.frc >> newDuff.frc; mv newDuff.frc DUFF2.frc\n\n")
147
148 def Runner():
149     if len(sys.argv) == 1:
150         NoArgRunner()
151     if len(sys.argv) > 1:
152         ArgRunner()
153
154 Runner()

```

APPENDIX C

Germanium Landscape

C.1 Sample Gaussian 09 Germanium File

Command files like the one below were built using Dr. Fennell's Gaussian 09 run builder script and proved very effective in producing command files.

```
1 #!/bin/bash
2 g09 <<EOF > B3LYP-STO-3G_1_hexagermane_transall_first_reorder.out
3 %Chk=B3LYP-STO-3G_1_hexagermane_transall_first_reorder
4 %NProcShared=12
5 #B3LYP/STO-3G OPT
6
7 Title: hexagermane_transall_first_reorder system
8
9 0 1
10 Ge      -4.543000000000      -0.076000000000      0.598000000000
11 Ge      -2.121000000000      0.068000000000      0.086000000000
12 C       -4.774000000000      -1.010000000000      2.336000000000
13 C       -5.549000000000      -1.127000000000      -0.760000000000
14 C       -5.371000000000      1.719000000000      0.829000000000
15 C       -1.311000000000      1.357000000000      1.303000000000
16 Ge      -0.754000000000      -1.978000000000      0.323000000000
17 C       -1.340000000000      -2.994000000000      1.865000000000
18 Ge       1.675000000000      -1.503000000000      0.567000000000
19 C      -1.186000000000      -3.039000000000      -1.235000000000
20 C       1.925000000000      -0.300000000000      2.094000000000
21 C       2.315000000000      -0.535000000000      -0.983000000000
22 Ge       3.126000000000      -3.539000000000      0.606000000000
23 C       2.522000000000      -4.845000000000      1.926000000000
24 Ge       5.558000000000      -3.333000000000      1.109000000000
25 C       2.938000000000      -4.327000000000      -1.151000000000
26 C       6.648000000000      -2.029000000000      0.052000000000
27 C       6.362000000000      -5.157000000000      0.995000000000
28 C       5.463000000000      -2.780000000000      3.007000000000
29 C      -4.808000000000      -2.522000000000      2.129000000000
30 C      -6.030000000000      -0.572000000000      3.098000000000
31 H      -3.909000000000      -0.757000000000      2.974000000000
32 H      -6.954000000000      -0.777000000000      2.525000000000
33 H      -5.990000000000      0.508000000000      3.335000000000
34 H      -6.094000000000      -1.114000000000      4.066000000000
35 H      -5.710000000000      -2.829000000000      1.564000000000
36 H      -4.817000000000      -3.048000000000      3.107000000000
37 H      -3.927000000000      -2.841000000000      1.555000000000
38 H      -5.250000000000      -2.187000000000      -0.668000000000
39 C      -5.199000000000      -0.705000000000      -2.186000000000
40 C      -7.061000000000      -1.057000000000      -0.533000000000
```

41	H	-4.126000000000	-0.879000000000	-2.361000000000
42	H	-5.433000000000	0.356000000000	-2.385000000000
43	H	-5.754000000000	-1.331000000000	-2.917000000000
44	H	-7.327000000000	-1.462000000000	0.459000000000
45	H	-7.588000000000	-1.677000000000	-1.290000000000
46	H	-7.438000000000	-0.019000000000	-0.608000000000
47	C	5.171000000000	-1.286000000000	3.108000000000
48	C	6.703000000000	-3.142000000000	3.823000000000
49	H	4.612000000000	-3.336000000000	3.456000000000
50	H	7.617000000000	-2.660000000000	3.425000000000
51	H	6.834000000000	-4.242000000000	3.835000000000
52	H	6.569000000000	-2.820000000000	4.877000000000
53	H	6.003000000000	-0.673000000000	2.715000000000
54	H	4.988000000000	-0.995000000000	4.164000000000
55	H	4.281000000000	-1.055000000000	2.511000000000
56	C	6.881000000000	-2.491000000000	-1.386000000000
57	C	8.000000000000	-1.692000000000	0.695000000000
58	H	6.093000000000	-1.084000000000	0.027000000000
59	H	8.663000000000	-2.569000000000	0.776000000000
60	H	7.871000000000	-1.257000000000	1.700000000000
61	H	8.527000000000	-0.927000000000	0.085000000000
62	H	7.531000000000	-3.384000000000	-1.416000000000
63	H	7.387000000000	-1.690000000000	-1.966000000000
64	H	5.929000000000	-2.720000000000	-1.888000000000
65	C	6.042000000000	-5.844000000000	-0.340000000000
66	H	5.912000000000	-5.747000000000	1.817000000000
67	C	7.882000000000	-5.177000000000	1.193000000000
68	H	8.397000000000	-4.688000000000	0.346000000000
69	H	8.240000000000	-6.228000000000	1.229000000000
70	H	8.189000000000	-4.690000000000	2.134000000000
71	H	4.959000000000	-5.905000000000	-0.514000000000
72	H	6.436000000000	-6.883000000000	-0.337000000000
73	H	6.487000000000	-5.311000000000	-1.199000000000
74	H	-6.362000000000	1.563000000000	1.303000000000
75	C	-5.646000000000	2.456000000000	-0.483000000000
76	C	-4.523000000000	2.590000000000	1.756000000000
77	H	-4.349000000000	2.080000000000	2.725000000000
78	H	-5.042000000000	3.550000000000	1.960000000000
79	H	-3.548000000000	2.821000000000	1.285000000000
80	H	-6.358000000000	1.894000000000	-1.110000000000
81	H	-4.725000000000	2.629000000000	-1.057000000000
82	H	-6.117000000000	3.440000000000	-0.273000000000
83	C	-0.532000000000	2.421000000000	0.817000000000
84	C	-1.529000000000	1.258000000000	2.684000000000
85	H	-2.129000000000	0.469000000000	3.088000000000
86	C	-0.996000000000	2.206000000000	3.561000000000
87	C	-0.001000000000	3.371000000000	1.694000000000
88	C	-0.237000000000	3.267000000000	3.066000000000
89	H	0.596000000000	4.188000000000	1.310000000000
90	H	-1.180000000000	2.122000000000	4.624000000000
91	H	0.174000000000	4.002000000000	3.745000000000
92	C	-1.777000000000	-4.322000000000	1.725000000000
93	C	-0.217000000000	-3.392000000000	-2.175000000000
94	C	-2.232000000000	-5.037000000000	2.838000000000

95	H	-1.775000000000	-4.812000000000	0.763000000000
96	C	-1.348000000000	-2.404000000000	3.134000000000
97	C	-0.568000000000	-4.133000000000	-3.309000000000
98	H	0.799000000000	-3.079000000000	-2.038000000000
99	C	-2.513000000000	-3.435000000000	-1.440000000000
100	C	-2.250000000000	-4.433000000000	4.097000000000
101	H	-2.571000000000	-6.058000000000	2.723000000000
102	C	-1.802000000000	-3.118000000000	4.246000000000
103	H	-1.007000000000	-1.394000000000	3.262000000000
104	C	-2.868000000000	-4.180000000000	-2.567000000000
105	H	-3.268000000000	-3.168000000000	-0.721000000000
106	C	-1.893000000000	-4.529000000000	-3.504000000000
107	H	0.183000000000	-4.395000000000	-4.040000000000
108	H	-3.896000000000	-4.482000000000	-2.715000000000
109	H	-2.164000000000	-5.101000000000	-4.381000000000
110	H	-2.602000000000	-4.985000000000	4.958000000000
111	H	-1.809000000000	-2.651000000000	5.222000000000
112	C	3.101000000000	-6.123000000000	1.997000000000
113	C	2.378000000000	-5.604000000000	-1.315000000000
114	C	3.370000000000	-3.619000000000	-2.281000000000
115	H	2.025000000000	-6.170000000000	-0.467000000000
116	C	2.272000000000	-6.169000000000	-2.590000000000
117	C	1.513000000000	-4.525000000000	2.832000000000
118	C	2.686000000000	-7.047000000000	2.960000000000
119	H	3.865000000000	-6.421000000000	1.310000000000
120	C	1.687000000000	-6.704000000000	3.869000000000
121	H	3.142000000000	-8.028000000000	3.002000000000
122	C	1.100000000000	-5.441000000000	3.804000000000
123	H	1.054000000000	-3.568000000000	2.784000000000
124	C	2.720000000000	-5.462000000000	-3.708000000000
125	H	1.844000000000	-7.156000000000	-2.710000000000
126	C	3.263000000000	-4.184000000000	-3.554000000000
127	H	3.780000000000	-2.628000000000	-2.178000000000
128	H	3.599000000000	-3.631000000000	-4.421000000000
129	H	2.636000000000	-5.900000000000	-4.694000000000
130	H	1.366000000000	-7.414000000000	4.620000000000
131	H	0.327000000000	-5.175000000000	4.510000000000
132	C	1.504000000000	-0.326000000000	-2.095000000000
133	C	1.670000000000	-0.714000000000	3.412000000000
134	C	3.620000000000	-0.040000000000	-0.992000000000
135	C	1.987000000000	0.362000000000	-3.212000000000
136	H	0.510000000000	-0.709000000000	-2.093000000000
137	C	2.402000000000	1.008000000000	1.889000000000
138	C	1.890000000000	0.147000000000	4.490000000000
139	H	1.314000000000	-1.696000000000	3.632000000000
140	C	4.114000000000	0.652000000000	-2.102000000000
141	H	4.233000000000	-0.179000000000	-0.124000000000
142	C	3.296000000000	0.851000000000	-3.216000000000
143	H	1.348000000000	0.514000000000	-4.073000000000
144	H	5.127000000000	1.034000000000	-2.095000000000
145	H	3.673000000000	1.385000000000	-4.079000000000
146	C	2.374000000000	1.435000000000	4.270000000000
147	H	1.691000000000	-0.189000000000	5.500000000000
148	C	2.630000000000	1.865000000000	2.969000000000

```

149 H      2.603000000000      1.384000000000      0.900000000000
150 H      3.001000000000      2.867000000000      2.795000000000
151 H      2.548000000000      2.101000000000      5.105000000000
152 C     -2.041000000000      0.841000000000     -1.709000000000
153 C     -1.767000000000      0.059000000000     -2.841000000000
154 C     -2.300000000000      2.209000000000     -1.888000000000
155 C     -1.732000000000      0.632000000000     -4.115000000000
156 H     -1.595000000000     -0.996000000000     -2.753000000000
157 C     -2.263000000000      2.785000000000     -3.160000000000
158 H     -2.521000000000      2.839000000000     -1.039000000000
159 C     -1.977000000000      1.997000000000     -4.275000000000
160 H     -1.519000000000      0.016000000000     -4.979000000000
161 H     -2.458000000000      3.843000000000     -3.281000000000
162 H     -1.950000000000      2.441000000000     -5.262000000000
163 H     -0.322000000000      2.526000000000     -0.236000000000
164
165 EOF
166 formchk B3LYP_STO-3G_1_hexagermane_transall_first_reorder.chk
167 newzmat -ichk -opdb -step 999 B3LYP_STO-3
      G_1_hexagermane_transall_first_reorder.chk final_B3LYP_STO-3
      G_1_hexagermane_transall_first_reorder.pdb
168 echo
169 echo "Job done"

```

C.2 Building Group 4 Chains

While briefly mentioned and the subject of research for some time, the butyl-IV chain builder is detailed below. Ultimately unsuccessful in the initial trials, these scripts may serve a purpose in further work.

This first script builds a parent set of all possible C, Si, and Ge butylalkyl chains.

```

1 #!/usr/bin/python
2
3 import sys
4 import subprocess
5
6 # argument: sys.argv[num]
7 # Replacement: sed -i -e 's/IN/OUT/g' FILE > NEWFILE
8
9 inFile = file(sys.argv[1])
10
11 def DoIT():
12     for first in {'C', 'Si', 'Ge'}:
13         name1 = "%s" % (first.lstrip(' '))
14         out1 = open(name1, "w")
15         cmdStr = "sed -e 's/1 GE/1 %s/g' ./%s >> ./%s.pdb" % (first,
16 inFile, name1)
17         # subprocess.call(cmdStr, shell=True, stdout=out1)
18         subprocess.Popen(cmdStr, shell=True, executable='/bin/bash')
19         out1.close()
20     for second in {'C', 'Si', 'Ge'}:
21         name2 = name1 + "_%s" % (second.lstrip(' '))
22         out2 = open(name2, "w")

```

```

22         cmdStr = "sed -e 's/2 GE/2 %s/g' ./%s.pdb >> ./%s.pdb" % (
second, name1, name2)
23         # subprocess.call(cmdStr, shell=True, stdout=out2)
24         subprocess.Popen(cmdStr, shell=True, executable='/bin/bash')
25         out2.close()
26         for third in {'C', 'Si', 'Ge'}:
27             name3 = name2 + "%s" % (third.lstrip(' '))
28             out3 = open(name3, "w")
29             cmdStr = "sed -e 's/3 GE/3 %s/g' ./%s.pdb >> ./%s.pdb" %
(third, name2, name3)
30             # subprocess.call(cmdStr, shell=True, stdout=out3)
31             subprocess.Popen(cmdStr, shell=True, executable='/bin/
bash')
32             out3.close()
33             for fourth in {'C', 'Si', 'Ge'}:
34                 name4 = name3 + "%s" % (fourth.lstrip(' '))
35                 out4 = open(name4, "w")
36                 cmdStr = "sed -e 's/4 GE/4 %s/g' ./%s.pdb >> ./%s.
pdb" % (fourth, name3, name4)
37                 # subprocess.call(cmdStr, shell=True, stdout=out4)
38                 subprocess.Popen(cmdStr, shell=True, executable='/
bin/bash')
39                 out4.close()
40
41 DoIT()

```

This second script takes the original trans-all butyl chain and enumerates 72 torsional rotations into a folder.

```

1 from chimera import runCommand as rc
2 from chimera import replyobj
3 import sys
4 import os
5
6 #standard sys.argv[] for script args?
7 # sys.argv[0] = directory
8 os.chdir(sys.argv[0])
9
10 file_names = [fn for fn in os.listdir(".") if fn.endswith(".pdb")]
11 fn = file_names[0]
12 # inPDB = chimera.openModels.open('/Users/gentry/Desktop/test/testmol.
pdb', type="PDB")
13
14 rc("open " + fn)
15
16 rc("rotation 1 reverse #0:1.HET@/serialNumber=2 #0:1.HET@/serialNumber=3
")
17
18 for i in range(72):
19     #replyobj.status("Processing " + fn)
20     #rc("open " + fn)
21     #rc("rotation 1 reverse #0:1.HET@/serialNumber=2 #0:1.HET@/
serialNumber=3")
22     rc("rotation 1 5")
23     newName = (fn[:-3] + str((i*5)) + ".pdb")

```

```

24     rc("write format pdb 0 " + newName)
25     #rc("close ")
26
27
28     # chimera.runCommand("rotation 2 3 5")
29     # newName = ( inPDB[:-3] + i*5 + ".pdb" )
30     # chimera.runCommand("write format pdb " + newName)

```

C.3 Collecting and Comparing Torsional Data

These two scripts were utilized to reduce the output data into an energy value with normalized intensity from 0 to 1. The third script compares two of these files and looks for any additive or multiplicative trend.

This first file reads energy data and creates a list of absolute energy values per torsion degree.

```

1  #!/usr/bin/python
2
3  #### Author: Gentry Smith, Oklahoma State University
4  #### Created: August 7, 2017, 3PM
5  #### Last Edited: August 7, 2017
6
7  #### Takes a stationary_points.txt file and will copy .pdb files of the
8  #### same name from a split_conformers.pdb/ folder
9  #### into a new folder "stationary_conformers"
10
11 # This does not use any args and instead relies on the stationary points
12 # file being "stationary_points.txt" and the
13 # conformers residing in a "split_conformers.pdb/" directory on the same
14 # level. It will create the new folder "stationary_conformers"
15
16 import os
17
18 def IOValidator():
19     returnBool = [False, False]
20     try:
21         file1 = open('stationary_points.txt', 'r')
22         file1.close()
23         returnBool[0] = True
24     except IOError:
25         print("Did not find 'stationary_points.txt' file. Quitting...")
26         quit()
27     try:
28         wkdir = os.getcwd()
29         file2 = os.chdir('split_conformers.pdb')
30         os.chdir(wkdir)
31         returnBool[1] = True
32     except OSError:
33         print("Did not find 'split_conformers.pdb' folder. Quitting...")
34         quit()
35     if returnBool[0] & returnBool[1]:
36         return True
37     else:

```

```

35         return False
36
37
38 def GetPDBs():
39     pdbNames = []
40     inFile = open('stationary_points.txt', 'r')
41     for line in inFile:
42         pdbNames.append(line.split()[1])
43     return pdbNames
44
45
46 def CopyPDBs(pdbList):
47     wkdir = os.getcwd()
48     for i in range(len(pdbList)):
49         pstring = ('cp' + 'split_conformers.pdb/' + str(pdbList[i]) +
50 'stationary_conformers/')
51         os.popen(pstring)
52
53 def Runner():
54     if IOValidator():
55         print('Valid Args. Running...')
56         pdbList = GetPDBs()
57         try:
58             os.mkdir('stationary_conformers')
59             CopyPDBs(pdbList)
60         except OSError:
61             print("'stationary_conformers' directory already exists.
62 Erase directory and run again. Quitting...")
63             quit()
64
65 Runner()

```

This second file converts the first file into a relative scale from 0 to 1.

```

1 #!/usr/bin/python
2
3 #### Author: Gentry Smith, Oklahoma State University
4 #### Created: July 31, 2017, 12PM
5 #### Last Edited: July 31, 2017
6
7 #### takes file arg with format [ [energy] [pdb_name] ], alters to [ [
8 energy] [torsion] ], and creates copy with
9 #### [ [relative energy] [torsion] ].
10
11
12 import sys
13
14 def IOValidator():
15     isValid = False
16     try:
17         inFile = sys.argv[1]
18         isValid = True
19     except IOError:

```



```

19         print("Input arg is not a file.\nQuitting...")
20         exit()
21     return isValid
22
23
24 def GetFileData():
25     inData = []
26     inFile = open(sys.argv[1], 'r')
27     iter = 0
28     for line in inFile:
29         inLine = line.split()
30         inData.append(float(inLine[0]))
31         iter = iter + 1
32     inFile.close()
33     return inData
34
35
36 def Relativize(energies):
37     minimum = min(energies)
38     # print("Relativize: minimum="+str(minimum))
39     newEnergies = []
40     for i in range(len(energies)):
41         # print("Relativize: index="+str(i))
42         # print("Relativize: energy="+str(energies[i]))
43         newMin = (float(energies[i]) - float(minimum))
44         # print("Relativize: newMin="+str(newMin))
45         newEnergies.append((newMin))
46         # print("Relativize: newEnergies="+str(newEnergies))
47     return newEnergies
48
49
50 def UnifiedScale(energies):
51     # print("unifying scale...")
52     maxi = max(energies)
53     # print("Unify: max=" + str(maxi))
54     newEnergies = []
55     for i in range(len(energies)):
56         # print("Unify: energy=" + str(energies[i]))
57         newEner = (float(energies[i]) / maxi)
58         # print("Unify: scaled energy=" + str(newEner))
59         newEnergies.append(newEner)
60     return newEnergies
61
62
63 def CriticalHit(energies, torsions):
64     isIncreasing = True
65     crits = []
66     tors = []
67     prev = 0
68     for i in range(len(energies)):
69         if (energies[i] == 0):
70             crits.append(energies[i])
71             tors.append(torsions[i])
72         if ((isIncreasing) & (energies[i] < prev)) or ((not

```

```

    isIncreasing) & (energies[i] > prev) ):
73         crits.append(energies[i-1])
74         tors.append(torsions[i-1])
75         isIncreasing = not isIncreasing
76         prev = float(energies[i])
77     returnThing = [crits, tors]
78     return returnThing
79
80
81 def MakeFile(energies, torsions, fileName):
82     outFile = open(fileName, 'w')
83     for i in range(len(energies)):
84         strOut = ('{:11e}'.format(energies[i]) + " " + str(torsions[i])
85 + "\n")
86         outFile.write(strOut)
87     outFile.close()
88
89 def Runner():
90     if IOValidator():
91         energies = GetFileData()
92         torsions = [180]
93         i = 185
94         while i != 180:
95             if i == 360:
96                 i = 0
97             torsions.append(i)
98             i = i + 5
99             MakeFile(energies, torsions, 'abs_energ.txt')
100             relativeEnergies = Relativize(energies)
101             MakeFile(relativeEnergies, torsions, 'rel_energ.txt')
102             MakeFile(UnifiedScale(relativeEnergies), torsions, 'uni_energ.
103 txt')
104             crits = CriticalHit(relativeEnergies, torsions)
105             MakeFile(crits[0], crits[1], 'crit_pts.txt')
106
107 Runner()

```

This third script compares two generated files using the prior scripts. It can compare the generated absolute energy with the relative energy files. It was often run as a loop through every permutation of the group 4 builder.

```

1 #!/usr/bin/python
2
3 ##### Author: Gentry Smith, Oklahoma State University
4 ##### Created: July 31, 2017, 3PM
5 ##### Last Edited: August 1, 2017
6
7 ##### Takes data created by teatAbsEnergies and compares values via
8 #####    additive and multiplicative comparison
9 #####    with abs or rel data. Math in terms of File 2 sub/div File 1.
10
11 # sys.argv[1] = file 1, working directory here.
12 # sys.argv[2] = file 2, compared with file 1.

```

```

12
13
14 import sys
15 import numpy
16 import math
17
18 def IOValidator():
19     isValid1 = False
20     isValid2 = False
21     try:
22         inFile1 = open(sys.argv[1])
23         isValid1 = True
24     except IOError:
25         print("Arg File 1 is invalid.")
26         isValid1 = False
27     try:
28         inFile1 = open(sys.argv[2])
29         isValid2 = True
30     except IOError:
31         print("Arg File 2 is invalid.")
32         isValid2 = False
33     if (isValid1 & isValid2 & (sys.argv[1] != sys.argv[2])):
34         print('Valid Args. Running...')
35         return True
36     else:
37         if (sys.argv[1] == sys.argv[2]):
38             print('args are identical. Skipping...')
39         else:
40             print("Invalid args. Quitting...")
41         exit()
42
43
44 def ExtractData(data):
45     inFile = open(data, 'r')
46     inData = []
47     inTorsions = []
48     # print('Extracting Data...')
49     for line in inFile:
50         # print('line=' + str(line))
51         # print('line.split()=' + str(line.split()))
52         # print('line.split()[1]=' + str(line.split()[1]))
53         inData.append(float(line.split()[0]))
54         inTorsions.append(int(line.split()[1]))
55     # print(str(inTorsions))
56     # print('Done.')
57     return [inData, inTorsions]
58
59
60 def Comparator(data1, data2, func):
61     # func: 0=add, 1=mult
62     newData = []
63     if func == 0:
64         for i in range(len(data2)):
65             newData.append(float(data2[i] - data1[i]))

```

```

66     elif func == 1:
67         for i in range(len(data2)):
68             try:
69                 newData.append(float(data2[i] / data1[i]))
70             except ZeroDivisionError:
71                 newData.append(0.0)
72     return newData
73
74
75 def WriteFile(data1, data2, tors, compData, comp, sigs):
76     # writes data of comparison. Format:
77     #   File1 = {file1}
78     #   File2 = {file2}
79     #   Source: {absolute, relative}
80     #   Comparison: {additive, multiplicative}
81     #   comp: {min/max/avg/stddev of all comp values}
82     #   Raw Data: {includes header of File1, File2, Torsions, Comp
83     # defining each column}
84     # print("Writing file...")
85     # print('File2=' + str((sys.argv[2]).split("/")))
86     source = ""
87     if str(sys.argv[1])[:3] == "abs":
88         source = "absolute"
89     elif str(sys.argv[1])[:3] == "rel":
90         source = "relative"
91     elif str(sys.argv[1])[:3] == "uni":
92         source = "unified relative scale"
93     else:
94         print(str(sys.argv[1])[:2])
95     comparison = ""
96     if comp == 0:
97         comparison = "additive"
98     elif comp == 1:
99         comparison = "multiplicative"
100     headerLines = [0]*10
101     headerLines[0] = ('File1 = ' + sys.argv[1] + '\n')
102     headerLines[1] = ('File2 = ' + sys.argv[2] + '\n')
103     headerLines[2] = ('Source: ' + source + '\n')
104     headerLines[3] = ('Comparison: ' + comparison + '\n')
105     headerLines[4] = ('Comparison min: ' + str(sigs[0]) + '\n')
106     headerLines[5] = ('Comparison max: ' + str(sigs[1]) + '\n')
107     headerLines[6] = ('Comparison avg: ' + str(sigs[2]) + '\n')
108     headerLines[7] = ('Comparison stddev: ' + str(sigs[3]) + '\n')
109     headerLines[8] = ('Raw Data: ' + '\n')
110     f1ColSize = len(str(data1[0]))
111     f2ColSize = len(str(data2[0]))
112     headerLines[9] = ('File1'.ljust(18) + 'File2'.ljust(18) + 'Tors'.
113     ljust(5) + 'Comp'.ljust(18) + '\n')
114     fileName = (str((sys.argv[2]).split("/"))[-2]) + "-" + str(sys.argv
115     [1])[:3] + "-" + comparison + '.txt')
116     outFile = open(fileName, 'w')
117     for i in range(len(headerLines)):
118         outFile.write(str(headerLines[i]))
119     for i in range(len(data1)):

```

```

117         # print('str(tors[i]).ljust(5)+' + str(tors[i]).ljust(5))
118         string = (str(data1[i])[:17].ljust(18) + ' ' + str(data2[i])
119                   [:17].ljust(18) + str(tors[i]).ljust(5) + str(compData[i])[:17].
120                   ljust(18) + '\n')
121         outFile.write(string)
122
123 def GetCompSigs(data):
124     sigs = []
125     sigs.append(min(data))
126     sigs.append(max(data))
127     sigs.append((float(sum(data))/float(len(data))))
128     sigs.append(numpy.std(data, axis=0))
129     return sigs
130
131 def Runner():
132     if IOValidator():
133         [data1, torsions1] = ExtractData(sys.argv[1])
134         [data2, torsions2] = ExtractData(sys.argv[2])
135         if (len(data1) == len(data2)) & (len(torsions1) == len(torsions2
136         )):
137             aData = Comparator(data1, data2, 0)
138             aSigs = GetCompSigs(aData)
139             WriteFile(data1, data2, torsions1, aData, 0, aSigs)
140             mData = Comparator(data1, data2, 1)
141             mSigs = GetCompSigs(mData)
142             WriteFile(data1, data2, torsions1, mData, 1, mSigs)
143             print('Complete.')
144 Runner()

```

APPENDIX D

Conformation Landscapes

Listed below are two example Germanium PDB files. The first is for the end-goal hexagermane in the trans-trans-trans conformation with isopropyl groups on the terminal Ge atoms. The second is for the simplified butagermane with fully protonated Germanium atoms.

D.1 Code: hexagermane-transall.pdb

```
1 HEADER
2 REMARK Title: hexagermane-transall system
3 HETATM 1 Ge 1 -4.399 0.008 0.355 0.00 0.00
   Ge
4 HETATM 2 Ge 1 -1.965 0.138 -0.022 0.00 0.00
   Ge
5 HETATM 3 C 1 -4.822 1.886 0.961 0.00 0.00
   C
6 HETATM 4 C 1 -5.008 -1.297 1.715 0.00 0.00
   C
7 HETATM 5 C 1 -5.256 -0.261 -1.445 0.00 0.00
   C
8 HETATM 6 C 1 -1.213 1.435 1.157 0.00 0.00
   C
9 HETATM 7 Ge 1 -0.756 -1.988 0.223 0.00 0.00
   Ge
10 HETATM 8 C 1 -1.297 -2.917 1.805 0.00 0.00
   C
11 HETATM 9 Ge 1 1.647 -1.496 0.371 0.00 0.00
   Ge
12 HETATM 10 C 1 -1.182 -3.010 -1.339 0.00 0.00
   C
13 HETATM 11 C 1 2.131 -0.425 1.877 0.00 0.00
   C
14 HETATM 12 C 1 2.111 -0.634 -1.269 0.00 0.00
   C
15 HETATM 13 Ge 1 2.889 -3.585 0.738 0.00 0.00
   Ge
16 HETATM 14 C 1 2.287 -4.358 2.378 0.00 0.00
   C
17 HETATM 15 Ge 1 5.327 -3.386 1.080 0.00 0.00
   Ge
18 HETATM 16 C 1 2.766 -4.685 -0.813 0.00 0.00
   C
19 HETATM 17 C 1 5.688 -2.615 2.887 0.00 0.00
   C
```

20	HETATM	18	C	1	6.239	-2.415	-0.417	0.00	0.00
		C							
21	HETATM	19	C	1	5.893	-5.324	0.888	0.00	0.00
		C							
22	HETATM	20	C	1	-3.527	2.543	1.328	0.00	0.00
		C							
23	HETATM	21	C	1	-5.754	1.844	2.133	0.00	0.00
		C							
24	HETATM	22	H	1	-5.303	2.355	0.072	0.00	0.00
		H							
25	HETATM	23	H	1	-5.269	1.358	2.999	0.00	0.00
		H							
26	HETATM	24	H	1	-6.679	1.287	1.913	0.00	0.00
		H							
27	HETATM	25	H	1	-6.047	2.856	2.449	0.00	0.00
		H							
28	HETATM	26	H	1	-3.043	2.019	2.171	0.00	0.00
		H							
29	HETATM	27	H	1	-3.683	3.585	1.642	0.00	0.00
		H							
30	HETATM	28	H	1	-2.818	2.559	0.490	0.00	0.00
		H							
31	HETATM	29	H	1	-4.336	-1.167	2.589	0.00	0.00
		H							
32	HETATM	30	C	1	-4.907	-2.680	1.154	0.00	0.00
		C							
33	HETATM	31	C	1	-6.417	-0.909	2.051	0.00	0.00
		C							
34	HETATM	32	H	1	-3.858	-2.985	0.982	0.00	0.00
		H							
35	HETATM	33	H	1	-5.433	-2.773	0.187	0.00	0.00
		H							
36	HETATM	34	H	1	-5.349	-3.420	1.836	0.00	0.00
		H							
37	HETATM	35	H	1	-6.488	0.167	2.302	0.00	0.00
		H							
38	HETATM	36	H	1	-6.802	-1.477	2.909	0.00	0.00
		H							
39	HETATM	37	H	1	-7.103	-1.094	1.205	0.00	0.00
		H							
40	HETATM	38	C	1	5.200	-6.127	1.944	0.00	0.00
		C							
41	HETATM	39	C	1	7.384	-5.453	0.966	0.00	0.00
		C							
42	HETATM	40	H	1	5.523	-5.590	-0.126	0.00	0.00
		H							
43	HETATM	41	H	1	7.790	-4.974	1.874	0.00	0.00
		H							
44	HETATM	42	H	1	7.885	-4.994	0.099	0.00	0.00
		H							
45	HETATM	43	H	1	7.691	-6.509	0.992	0.00	0.00
		H							
46	HETATM	44	H	1	5.502	-5.821	2.960	0.00	0.00
		H							

47	HETATM	45	H	1	5.436	-7.197	1.849	0.00	0.00
		H							
48	HETATM	46	H	1	4.106	-6.027	1.879	0.00	0.00
		H							
49	HETATM	47	C	1	6.243	-1.232	2.746	0.00	0.00
		C							
50	HETATM	48	C	1	6.612	-3.524	3.636	0.00	0.00
		C							
51	HETATM	49	H	1	4.684	-2.582	3.376	0.00	0.00
		H							
52	HETATM	50	H	1	7.535	-3.731	3.068	0.00	0.00
		H							
53	HETATM	51	H	1	6.139	-4.497	3.853	0.00	0.00
		H							
54	HETATM	52	H	1	6.913	-3.088	4.599	0.00	0.00
		H							
55	HETATM	53	H	1	7.243	-1.234	2.279	0.00	0.00
		H							
56	HETATM	54	H	1	6.347	-0.742	3.725	0.00	0.00
		H							
57	HETATM	55	H	1	5.589	-0.589	2.128	0.00	0.00
		H							
58	HETATM	56	C	1	5.630	-1.055	-0.555	0.00	0.00
		C							
59	HETATM	57	H	1	6.024	-3.039	-1.315	0.00	0.00
		H							
60	HETATM	58	C	1	7.712	-2.342	-0.145	0.00	0.00
		C							
61	HETATM	59	H	1	7.923	-1.890	0.839	0.00	0.00
		H							
62	HETATM	60	H	1	8.227	-1.728	-0.898	0.00	0.00
		H							
63	HETATM	61	H	1	8.188	-3.335	-0.163	0.00	0.00
		H							
64	HETATM	62	H	1	4.573	-1.106	-0.861	0.00	0.00
		H							
65	HETATM	63	H	1	6.155	-0.455	-1.314	0.00	0.00
		H							
66	HETATM	64	H	1	5.675	-0.486	0.391	0.00	0.00
		H							
67	HETATM	65	H	1	-5.890	-1.163	-1.302	0.00	0.00
		H							
68	HETATM	66	C	1	-4.220	-0.487	-2.505	0.00	0.00
		C							
69	HETATM	67	C	1	-6.093	0.945	-1.729	0.00	0.00
		C							
70	HETATM	68	H	1	-6.841	1.122	-0.939	0.00	0.00
		H							
71	HETATM	69	H	1	-6.644	0.838	-2.676	0.00	0.00
		H							
72	HETATM	70	H	1	-5.478	1.858	-1.818	0.00	0.00
		H							
73	HETATM	71	H	1	-3.754	-1.481	-2.414	0.00	0.00
		H							

74	HETATM	72	H	1	-3.411	0.262	-2.459	0.00	0.00
		H							
75	HETATM	73	H	1	-4.659	-0.429	-3.512	0.00	0.00
		H							
76	HETATM	74	C	1	-1.706	1.681	2.429	0.00	0.00
		C							
77	HETATM	75	C	1	-0.128	2.155	0.679	0.00	0.00
		C							
78	HETATM	76	H	1	0.268	1.941	-0.323	0.00	0.00
		H							
79	HETATM	77	C	1	0.451	3.147	1.465	0.00	0.00
		C							
80	HETATM	78	C	1	-1.134	2.678	3.216	0.00	0.00
		C							
81	HETATM	79	C	1	-0.058	3.415	2.731	0.00	0.00
		C							
82	HETATM	80	H	1	-1.525	2.873	4.219	0.00	0.00
		H							
83	HETATM	81	H	1	1.306	3.716	1.086	0.00	0.00
		H							
84	HETATM	82	H	1	0.391	4.199	3.349	0.00	0.00
		H							
85	HETATM	83	C	1	-1.557	-4.274	1.694	0.00	0.00
		C							
86	HETATM	84	C	1	-0.365	-3.088	-2.455	0.00	0.00
		C							
87	HETATM	85	C	1	-2.027	-4.985	2.796	0.00	0.00
		C							
88	HETATM	86	H	1	-1.378	-4.792	0.742	0.00	0.00
		H							
89	HETATM	87	C	1	-1.446	-2.277	3.025	0.00	0.00
		C							
90	HETATM	88	C	1	-0.752	-3.866	-3.544	0.00	0.00
		C							
91	HETATM	89	H	1	0.592	-2.548	-2.482	0.00	0.00
		H							
92	HETATM	90	C	1	-2.386	-3.699	-1.304	0.00	0.00
		C							
93	HETATM	91	C	1	-2.219	-4.336	4.011	0.00	0.00
		C							
94	HETATM	92	H	1	-2.237	-6.056	2.707	0.00	0.00
		H							
95	HETATM	93	C	1	-1.915	-2.983	4.130	0.00	0.00
		C							
96	HETATM	94	H	1	-1.159	-1.217	3.132	0.00	0.00
		H							
97	HETATM	95	C	1	-2.771	-4.484	-2.388	0.00	0.00
		C							
98	HETATM	96	H	1	-3.043	-3.610	-0.422	0.00	0.00
		H							
99	HETATM	97	C	1	-1.952	-4.568	-3.509	0.00	0.00
		C							
100	HETATM	98	H	1	-0.105	-3.928	-4.425	0.00	0.00
		H							

101	HETATM	99	H	1	-3.721	-5.027	-2.358	0.00	0.00
		H							
102	HETATM	100	H	1	-2.253	-5.182	-4.364	0.00	0.00
		H							
103	HETATM	101	H	1	-2.596	-4.891	4.876	0.00	0.00
		H							
104	HETATM	102	H	1	-2.041	-2.474	5.091	0.00	0.00
		H							
105	HETATM	103	C	1	2.487	-3.679	3.571	0.00	0.00
		C							
106	HETATM	104	C	1	1.701	-5.563	-0.935	0.00	0.00
		C							
107	HETATM	105	C	1	3.733	-4.618	-1.807	0.00	0.00
		C							
108	HETATM	106	H	1	0.940	-5.615	-0.140	0.00	0.00
		H							
109	HETATM	107	C	1	1.598	-6.382	-2.057	0.00	0.00
		C							
110	HETATM	108	C	1	1.690	-5.609	2.382	0.00	0.00
		C							
111	HETATM	109	C	1	2.102	-4.259	4.776	0.00	0.00
		C							
112	HETATM	110	H	1	2.956	-2.680	3.567	0.00	0.00
		H							
113	HETATM	111	C	1	1.520	-5.523	4.784	0.00	0.00
		C							
114	HETATM	112	H	1	2.260	-3.721	5.716	0.00	0.00
		H							
115	HETATM	113	C	1	1.311	-6.197	3.585	0.00	0.00
		C							
116	HETATM	114	H	1	1.504	-6.131	1.431	0.00	0.00
		H							
117	HETATM	115	C	1	2.562	-6.313	-3.057	0.00	0.00
		C							
118	HETATM	116	H	1	0.754	-7.074	-2.153	0.00	0.00
		H							
119	HETATM	117	C	1	3.630	-5.430	-2.933	0.00	0.00
		C							
120	HETATM	118	H	1	4.590	-3.931	-1.700	0.00	0.00
		H							
121	HETATM	119	H	1	4.391	-5.376	-3.718	0.00	0.00
		H							
122	HETATM	120	H	1	2.481	-6.954	-3.941	0.00	0.00
		H							
123	HETATM	121	H	1	1.223	-5.984	5.731	0.00	0.00
		H							
124	HETATM	122	H	1	0.844	-7.187	3.587	0.00	0.00
		H							
125	HETATM	123	C	1	1.878	0.732	-1.306	0.00	0.00
		C							
126	HETATM	124	C	1	1.530	-0.534	3.120	0.00	0.00
		C							
127	HETATM	125	C	1	2.642	-1.289	-2.370	0.00	0.00
		C							

128	HETATM	126	C	1	2.179	1.455	-2.458	0.00	0.00
		C							
129	HETATM	127	H	1	1.444	1.239	-0.432	0.00	0.00
		H							
130	HETATM	128	C	1	3.179	0.461	1.679	0.00	0.00
		C							
131	HETATM	129	C	1	2.005	0.227	4.186	0.00	0.00
		C							
132	HETATM	130	H	1	0.661	-1.197	3.265	0.00	0.00
		H							
133	HETATM	131	C	1	2.940	-0.568	-3.524	0.00	0.00
		C							
134	HETATM	132	H	1	2.840	-2.370	-2.334	0.00	0.00
		H							
135	HETATM	133	C	1	2.710	0.804	-3.567	0.00	0.00
		C							
136	HETATM	134	H	1	1.989	2.533	-2.491	0.00	0.00
		H							
137	HETATM	135	H	1	3.358	-1.081	-4.396	0.00	0.00
		H							
138	HETATM	136	H	1	2.944	1.370	-4.475	0.00	0.00
		H							
139	HETATM	137	C	1	3.067	1.105	3.998	0.00	0.00
		C							
140	HETATM	138	H	1	1.534	0.140	5.170	0.00	0.00
		H							
141	HETATM	139	C	1	3.650	1.229	2.740	0.00	0.00
		C							
142	HETATM	140	H	1	3.633	0.553	0.682	0.00	0.00
		H							
143	HETATM	141	H	1	4.480	1.926	2.585	0.00	0.00
		H							
144	HETATM	142	H	1	3.439	1.703	4.836	0.00	0.00
		H							
145	HETATM	143	C	1	-2.039	0.838	-1.804	0.00	0.00
		C							
146	HETATM	144	C	1	-1.525	0.195	-2.916	0.00	0.00
		C							
147	HETATM	145	C	1	-2.655	2.077	-1.927	0.00	0.00
		C							
148	HETATM	146	C	1	-1.618	0.802	-4.168	0.00	0.00
		C							
149	HETATM	147	H	1	-1.048	-0.789	-2.818	0.00	0.00
		H							
150	HETATM	148	C	1	-2.746	2.686	-3.175	0.00	0.00
		C							
151	HETATM	149	H	1	-3.084	2.566	-1.036	0.00	0.00
		H							
152	HETATM	150	C	1	-2.223	2.047	-4.296	0.00	0.00
		C							
153	HETATM	151	H	1	-1.210	0.296	-5.049	0.00	0.00
		H							
154	HETATM	152	H	1	-3.229	3.663	-3.275	0.00	0.00
		H							

155	HETATM	153	H		1		-2.292	2.524	-5.279	0.00	0.00
		H									
156	HETATM	154	H		1		-2.539	1.081	2.827	0.00	0.00
		H									
157	CONECT	3	1	20	21	22					
158	CONECT	4	1	29	30	31					
159	CONECT	5	1	65	66	67					
160	CONECT	6	74	75	2						
161	CONECT	8	83	87	7						
162	CONECT	10	84	90	7						
163	CONECT	11	124	128	9						
164	CONECT	12	123	125	9						
165	CONECT	14	103	108	13						
166	CONECT	16	104	105	13						
167	CONECT	17	15	47	48	49					
168	CONECT	18	56	57	58	15					
169	CONECT	19	15	38	39	40					
170	CONECT	20	3	26	27	28					
171	CONECT	21	3	23	24	25					
172	CONECT	30	4	34	32	33					
173	CONECT	31	4	35	36	37					
174	CONECT	38	19	44	45	46					
175	CONECT	39	19	41	42	43					
176	CONECT	47	54	55	17	53					
177	CONECT	48	17	50	51	52					
178	CONECT	56	62	63	64	18					
179	CONECT	58	59	60	61	18					
180	CONECT	66	71	72	73	5					
181	CONECT	67	68	69	70	5					
182	CONECT	74	78	154	6						
183	CONECT	75	76	77	6						
184	CONECT	77	75	79	81						
185	CONECT	78	74	79	80						
186	CONECT	79	77	78	82						
187	CONECT	83	85	86	8						
188	CONECT	84	88	89	10						
189	CONECT	85	83	91	92						
190	CONECT	87	93	94	8						
191	CONECT	88	84	97	98						
192	CONECT	90	95	96	10						
193	CONECT	91	85	93	101						
194	CONECT	93	87	91	102						
195	CONECT	95	90	97	99						
196	CONECT	97	88	95	100						
197	CONECT	103	109	110	14						
198	CONECT	104	106	107	16						
199	CONECT	105	117	118	16						
200	CONECT	107	104	115	116						
201	CONECT	108	113	114	14						
202	CONECT	109	103	111	112						
203	CONECT	111	109	113	121						
204	CONECT	113	108	111	122						
205	CONECT	115	107	117	120						
206	CONECT	117	105	115	119						

207	CONECT	123	126	127	12
208	CONECT	124	129	130	11
209	CONECT	125	131	132	12
210	CONECT	126	123	133	134
211	CONECT	128	139	140	11
212	CONECT	129	124	137	138
213	CONECT	131	125	133	135
214	CONECT	133	126	131	136
215	CONECT	137	129	139	142
216	CONECT	139	128	137	141
217	CONECT	143	144	145	2
218	CONECT	144	143	146	147
219	CONECT	145	143	148	149
220	CONECT	146	144	150	151
221	CONECT	148	145	150	152
222	CONECT	150	146	148	153
223	CONECT	1	2	3	4
224	CONECT	2	1	143	6
225	CONECT	7	2	8	9
226	CONECT	9	7	11	12
227	CONECT	13	9	14	15
228	CONECT	15	13	17	18
229	CONECT	22	3		
230	CONECT	23	21		
231	CONECT	24	21		
232	CONECT	25	21		
233	CONECT	26	20		
234	CONECT	27	20		
235	CONECT	28	20		
236	CONECT	29	4		
237	CONECT	32	30		
238	CONECT	33	30		
239	CONECT	34	30		
240	CONECT	35	31		
241	CONECT	36	31		
242	CONECT	37	31		
243	CONECT	40	19		
244	CONECT	41	39		
245	CONECT	42	39		
246	CONECT	43	39		
247	CONECT	44	38		
248	CONECT	45	38		
249	CONECT	46	38		
250	CONECT	49	17		
251	CONECT	50	48		
252	CONECT	51	48		
253	CONECT	52	48		
254	CONECT	53	47		
255	CONECT	54	47		
256	CONECT	55	47		
257	CONECT	57	18		
258	CONECT	59	58		
259	CONECT	60	58		
260	CONECT	61	58		

```

261 CONECT    62    56
262 CONECT    63    56
263 CONECT    64    56
264 CONECT    65     5
265 CONECT    68    67
266 CONECT    69    67
267 CONECT    70    67
268 CONECT    71    66
269 CONECT    72    66
270 CONECT    73    66
271 CONECT    76    75
272 CONECT    80    78
273 CONECT    81    77
274 CONECT    82    79
275 CONECT    86    83
276 CONECT    89    84
277 CONECT    92    85
278 CONECT    94    87
279 CONECT    96    90
280 CONECT    98    88
281 CONECT    99    95
282 CONECT   100    97
283 CONECT   101    91
284 CONECT   102    93
285 CONECT   106   104
286 CONECT   110   103
287 CONECT   112   109
288 CONECT   114   108
289 CONECT   116   107
290 CONECT   118   105
291 CONECT   119   117
292 CONECT   120   115
293 CONECT   121   111
294 CONECT   122   113
295 CONECT   127   123
296 CONECT   130   124
297 CONECT   132   125
298 CONECT   134   126
299 CONECT   135   131
300 CONECT   136   133
301 CONECT   138   129
302 CONECT   140   128
303 CONECT   141   139
304 CONECT   142   137
305 CONECT   147   144
306 CONECT   149   145
307 CONECT   151   146
308 CONECT   152   148
309 CONECT   153   150
310 CONECT   154    74
311 END

```

The above molecule contains 154 atoms and 153 bonds, making it extremely computationally expensive for regular QM calculations. This made utilizing the large

molecule as a trial system unreasonable due to the prohibitively long computation time for each conformation, assuming the conformation calculation would complete at all.

The below PDB file is the simplified butagermane with fully protonated Germanium atoms. As a significantly smaller system with only 14 atoms and 13 bonds, the relatively short computation time allowed the trial system to move with relative ease.

D.2 Code: ge4h.pdb

```

1 COMPND      UNNAMED
2 AUTHOR      GENERATED BY OPEN BABEL 2.3.90
3 HETATM      1 GE      UNL      1      -3.520      1.842      -0.078      1.00      0.00
               Ge3—
4 HETATM      2 GE      UNL      1      -1.368      2.888      -0.034      1.00      0.00
               Ge2—
5 HETATM      3 GE      UNL      1      0.324      1.200      0.059      1.00      0.00
               Ge3—
6 HETATM      4 GE      UNL      1      2.475      2.248      0.099      1.00      0.00
               Ge
7 HETATM      5 H      UNL      1      -4.622      2.930      -0.135      1.00      0.00
               H
8 HETATM      6 H      UNL      1      -3.699      0.985      1.202      1.00      0.00
               H
9 HETATM      7 H      UNL      1      -3.621      0.932      -1.328      1.00      0.00
               H
10 HETATM     8 H      UNL      1      -1.258      3.797      1.217      1.00      0.00
               H
11 HETATM     9 H      UNL      1      -1.178      3.740      -1.314      1.00      0.00
               H
12 HETATM    10 H      UNL      1      0.213      0.288      -1.189      1.00      0.00
               H
13 HETATM    11 H      UNL      1      0.135      0.352      1.342      1.00      0.00
               H
14 HETATM    12 H      UNL      1      2.655      3.095      -1.186      1.00      0.00
               H
15 HETATM    13 H      UNL      1      3.578      1.161      0.165      1.00      0.00
               H
16 HETATM    14 H      UNL      1      2.574      3.167      1.343      1.00      0.00
               H
17 CONECT      1      2      5      6      7
18 CONECT      2      1      3      8      9
19 CONECT      3      2      4     10     11
20 CONECT      4      3     12     13     14
21 CONECT      5      1
22 CONECT      6      1
23 CONECT      7      1
24 CONECT      8      2
25 CONECT      9      2
26 CONECT     10      3
27 CONECT     11      3
28 CONECT     12      4
29 CONECT     13      4
30 CONECT     14      4

```

```

31 MASTER          0    0    0    0    0    0    0    0    14    0    14    0
32 END

```

D.3 Progress on Torsion Minimizer System

While incomplete and largely nonfunctioning, this code is the current progress toward the implementation of the torsion minimizer system as outlined in 5.2.

```

1  ##### Author: Gentry Smith
2  ##### Date: April 22, 2017
3  ##### Description: This is the runner file that is the primary executable
4  #####                for the torsion minimizer. Currently is the
5  #####                only file utilized.
6
7  # Inputs:
8  # Arg1: the molecule file to be minimized (currently only accepts a pdb
9  #       file)
10
11 import sys
12 import subprocess
13 import math
14
15 # IO Validator: validates user-submitted molecule.
16 def IOValidator():
17     isValid = False
18     # Check for valid length of args (2)
19     if len(sys.argv) == 2:
20         # Check arg to make sure it's a file.
21         argFile = sys.argv[1]
22         try:
23             inputFile = open(argFile)
24             # Finally, make sure the file is a .pdb
25             if inputFile[-4:] == ".pdb":
26                 isValid = True
27             else:
28                 print("This is not a .pdb file. Please try again with a
29                 .pdb file.\n")
30             inputFile.close()
31         except IOError:
32             print("System was not able to open '", str(argFile), "'.")
33     # too long
34     elif len(sys.argv) > 2:
35         print("You have too many arguments. Call the file as 'Runner.py
36         [molecule file]' and try again.\n")
37     # too short
38     else:
39         print("You do not have enough arguments. Start the program as '
40         Runner.py [molecule file]' and try again.\n")
41     # return validity boolean
42     return isValid
43
44 # Get Torsions: initiates function to get user-specified torsion bonds.
45 # Returns bonds as int [[a,b],[a,b]] list
46 def getTorsions():
47     torsions = [[0, 0]]

```



```

42 newTorsion = "first"
43 firstTime = True
44 doneCheck = ""
45 badIn = False
46
47 # loop for all torsions until user types "done"
48 while newTorsion != "":
49     if firstTime:
50         print("It's time to define the torsions of the molecule and
51         declare which bonds you would like to rotate.\n")
52         print("Before going any further, it's important to note at
53         this time that version 0.2 (current) will assume the torsions you
54         enter are completely correct. You'll see a bunch of error messages
55         soon if it isn't correct.\n")
56         print("Open the .pdb file and identify the numbers of the
57         atoms on the .pdb that will make the bond (the first number on the
58         line of each atom)\n\n")
59         print("Now it's time to enter in the numbers of the two
60         atoms. We'll do it one at a time.")
61
62         firstTor = raw_input("Type in the number of the first atom
63         in the bond and hit enter. \nEx: type 3 and then hit enter.\n")
64
65         try:
66             confFirstTor = int(firstTor)
67         except ValueError:
68             print("You typed in '", firstTor, "'", which is not a
69             number. Let's start again.")
70             badIn = True
71
72         secondTor = raw_input("Type in the number of the second atom
73         in the bond and hit enter. \nEx: type 3 and then hit enter.\n")
74
75         try:
76             confSecondTor = int(secondTor)
77         except ValueError:
78             print("You typed in '", secondTor, "'", which is not a
79             number. Let's start again.")
80             badIn = True
81             firstTime = False
82
83     else:
84         print("Open the .pdb file and identify the numbers of the
85         atoms on the .pdb that will make the bond (the first number on the
86         line of each atom)\n\n")
87
88         firstTor = raw_input("Type in the number of the first atom
89         in the bond and hit enter. \nEx: type 3 and then hit enter.\n")
90
91         try:
92             confFirstTor = int(firstTor)
93         except ValueError:
94             print("You typed in '", firstTor, "'", which is not a
95             number. Let's start again.")

```

```

81         badIn = True
82
83         secondTor = raw_input("Type in the number of the second atom
in the bond and hit enter. \nEx: type 3 and then hit enter.\n")
84
85         try:
86             confSecondTor = int(secondTor)
87         except ValueError:
88             print("You typed in '", secondTor, "'", which is not a
number. Let's start again.")
89             badIn = True
90             firstTime = False
91
92     if badIn == False:
93         newTorsion = [confFirstTor, confSecondTor]
94         if torsions == [[0, 0]]:
95             print("You added a new torsion: ", newTorsion, "\n")
96             torsions = newTorsion
97         else:
98             torsions.append(newTorsion)
99             print("The current torsions you have created are:\n")
100             for each in torsions:
101                 print(each, "\n")
102             doneCheck = raw_input("If you would like to add another
torsion, press enter. If you are finished adding torsions, type '
done' and press enter\n")
103
104             if str(doneCheck) == "done":
105                 print("Finished entering torsions. Begining the work.\n")
106             )
107             else:
108                 newTorsion = "first"
109
110     if badIn == True:
111         firstTime = True
112         badIn = False
113         newTorsion = "first"
114
115     return torsions
116
117 # Get Conformation Count: determines conformations needed. Returns list
in form: [#conf, rotDeg, rotRng]
118 def getConformationInfo(depth, torsions):
119     # rotates 60 degrees on the first search, then logarithmic decrease
from 10 for each subsequent search.
120     rotDeg = [60, 10]
121     # full torsion range for first search, logarithmic decrease from 50
for each subsequent search
122     rotRng = [360, 50]
123     # number of conformations needed
124     numConf = 0
125     # degrees per rotation
126     deg = 0
127     # rotation range

```

```

127     rng = 0
128     # number of rotations per torsion
129     rotTick = 0
130
131     # determine counts from depth
132     if depth >= 2:
133         deg = math.pow(10, (2-depth))
134         rng = deg*5
135     elif depth <2:
136         deg = rotDeg[depth]
137         rng = rotRng[depth]
138     if depth == 1:
139         rotTick = 6
140     elif depth >= 1:
141         rotTick = 11
142
143     numConf = math.pow(torsions, rotTick)
144
145     return [numConf, deg, rng]
146
147
148
149
150
151
152
153
154 def Launcher():
155     valid = IOValidator()
156     if valid:
157         # do everything
158         depth = 0
159
160         InitWD()
161
162
163     else:
164         print("There was a problem while reading in the molecule file.
165         Please try again.\n")
166         exit()
167
168 # Initiates proper working directory.
169 def InitWD():
170
171
172
173
174 # Recursive search through molecule torsions
175 def RecursiveSearch(depth):
176
177     torsions = getTorsions()
178
179

```

180

181 **Launcher()**

VITA

Gentry H. Smith

Candidate for the Degree of
Master of Science

Thesis: EXPLORING CRITICAL CONFORMATIONS

Major Field: Chemistry

Biographical:

Personal Information: Born in Olathe, KS in November 1993.

Education:

Received a Bachelors of Science in Chemistry at Southern Nazarene University in May 2016.

Completed the requirements for the degree of Master of Science with a major in Chemistry at Oklahoma State University in November 2018.

Experience:

Teaching Assistant, various undergraduate chemistry courses, Southern Nazarene University, Aug. 2014 - May 2016

Graduate Teaching Assistant, CHEM 1314, 1414 at Oklahoma State University, Aug. 2016 - Dec. 2018

Professional Affiliations:

American Chemical Society

Awards

Colonel Andre Whitely Scholarship in Chemistry