CHAPTER 1

Introduction

1.1 Computational Chemistry: Chemistry on the Computer

For nearly a century, computational methods have greatly assisted chemists in their efforts of research and discovery and have relatively recently become their own field of focus within chemistry. As a standalone field, computational chemistry uses computer simulations to solve chemical problems. These simulations typically rely on theoretical methods adapted to run highly efficiently on computers. While initial computational methods were designed to solve wave functions and atomic orbitals, the scope quickly expanded into multiple fields of chemistry as more methods were developed to confirm or predict properties of molecules and systems.

With the introduction of *ab initio* and density functional methods, computational methods began to stand as a distinct field within chemistry. This introduction serves to introduce necessary background information generally relevant to the methods developed and utilized in the following chapters.

1.2 Relevant Computational Methods

Modern computational methods take one of several approaches in computing a system. Usually limited by the scale of molecule and scope of system, multiple methods exist to analytically solve or closely approximate a system by way of solving or approximating the quantum mechanical wave function or through a similarly consistent method of generalization by way of statistical solutions. Many other methods exist, but are not directly relevant to this work.

The first hurdle in any computational system is the likely impossibility of analytically solving the problem. In a system with more than two particles, this multi-body problem usually cannot be solved analytically excepting the dihydrogen cation due to the exorbitant amount of information contained in solving the wave function. This is largely due to the complexity introduced into the wave function of multiple interacting species that often lead to prohibitively large or impossible equations to solve.

1.2.1 Quantum Mechanical Methods and Basis Sets

In computational chemistry, quantum mechanical methods generally refer to computational methods that attempt to solve or closely approximate the electronic Schrdinger equation given nuclei and electron position information to determine properties of the system like energies or electron densities. Because the Schrdinger equation is impossible to solve for many-body systems, different methods use different approximations to balance between accuracy of the approximation and efficiency of computation.

1.2.1.1 Basis Sets

While running calculations, both *Ab initio* and DFT methods require basis sets to represent the electronic wave function as a system of algebraic equations that can be efficiently calculated. While basis sets can be designed of atomic orbitals or plane waves, this work focus primarily on basis sets derived by atomic orbitals. The two most often used types of orbitals are Gaussian-type and Slater-type orbitals. Slater-type orbitals (STOs), named after the physicist John Slater who introduced them in 1930,[?] function as a linear combination of atomic orbitals (LCAO) adopted as a molecular orbital. STOs notably exhibit similar properties to Schrdinger-based orbitals, excepting that STOs have no radial nodes.

Gaussian-type orbitals (GTOs), introduced by S. Francis Boys in 1950,[?] also function as orbitals in the LCAO method. GTOs are similar to STOs in premise, but have further reduced realism when compared to Schrdinger-based orbitals. One example of this is the lack of accuracy of electron density near the nucleus. While exhibiting a lesser accuracy, GTOs excel in computational efficiency compared to STOs by roughly 4-5 orders of magnitude. This allows GTO-based calculations to compute more orbitals. Specifically, Boys designed GTOs as a method of approximating STOs.

Basis sets are often grouped by their sizes. The smallest sets, known as minimal basis sets, use a single basis function for each orbital. The most common minimal basis set, STO-nG where n is an integer usually between 2 and 6, was first proposed by John Pople in 1969.[?] This method describes that a Slater-type orbital can be approximated using n-Gaussian orbitals. These STO-nG approximations end up fitting electron densities well at all radial distances except those close to the nucleus. The STO-3G basis set used in this work is a popular basis set as the 3 Gaussian-type orbitals works well for atoms in the [H-Xe] range.

The other basis sets used in the work fall under the category of split-valence basis sets. These basis sets represent valence electrons with more than one basis function, which allows for electron density to be more flexible in different molecular systems. The most common form of these basis sets were introduced by John Pople as the X-YZg form and are commonly referred to as Pople basis sets.[?] These follow the form that each orbital basis function is comprised of X Gaussians. The Y and Z represent the two additional linear combination of Gaussian functions made of Y and Z Gaussians each, respectively, that compose the valence. These basis sets are not limited to two valence functions, referred to as a double-zeta, and can also be triple- or quadruple-zeta. Additional values, typically denoted by one or two stars, one or two plus signs, or explitly-defined orbital combinations in parentheses can also be used to further complicate the basis set as desired. The star denotion defines a polarization function for heavy atoms to account for d and f polarizations. The plus signs denote diffuse functions that more-accurately represent less common valence electrons like carbanions that may diffuse further out from the nucleus.

1.2.1.2 *Ab Initio* Methods

Ab initio, or "from first principles," methods refer to calculation methods that rely solely on physical constants as external values. By design, *ab initio* methods avoid using any empirically-acquired data and rely on theoretically calculated values. The development of these methods allowed computational chemists to solve a new class of problems and resulted in John Pople and Walter Kohn receiving the Nobel Prize for their work. The *ab initio* method utilized in this work is the Hartree-Fock (HF) method used to determine the energy of a many-body system in a stationary state, which is to say time-independent.[?] Known initially as the self-consistent field method, the HF method utilizes approximations defined by the basis set to approximate the Schrdinger equation. The consistency of this self-consistent field method arose by the requirement that the final calculated field be self-consistent with the initial field. An additional property of HF is that electron-electron repulsion is not taken into account, requiring that a basis set account for this interaction. As larger basis sets are used, the overall energy of the wavefunction is increased toward a value known as the Hartree-Fock limit. This limit is approached as the larger basis sets approach the exact solution of the non-relativistic Schrdinger equation without spin orbital terms. The calculation of relativistic and spin terms require a further method known as Post-Hartree-Fock, which is not used considered further in this work.

1.2.1.3 Density Functional Theory Methods

Density Function Theory (DFT) Methods function very similarly to *ab initio* methods in how Slater-type orbitals are used to approximate the Schrdinger equation, but differ in that DFT utilizes some empirical data to speed up the calculation process.[?] These simplifications are stable able to model exchange and correlation interactions very well, however the reliability of calculated properties, specifically intermolecular interactions, dispersion forces, and other internal properties are greatly reduced. Just as with *ab initio* methods, DFT methods require a basis set definition for the approximation calculations. Many DFT methods exist and even some so-called hybrid functional methods that exchange with HF terms for greater reliability in calculated values. One pure DFT method used in this work is BLYP, which utilizes the Becke exchange with the Lee-Yang-Parr correlation part. Some hybrid functional methods used are the B3LYP, M06L, and PBE methods. The B3LYP utilizes the BLYP but combined Becke's exchange with the exact energy from HF theory. M06L, known as the Minnesota functionals, depend on kinetic energy density values from databases. It specifically was designed to work well with transition metals, inorganics, and organometallics.[?] The PBE method is another method with similar levels of accuracy to B3LYP that attempts to increase the number of HF-exchanged functionals.

1.2.1.4 Semi-Empirical Methods

Like DFT, Semi-empirical methods also pull somewhat from Hartree-Fock methods, but rely even more on approximations and empirical data to nearly completely substitute out any proper calculation of the Schrdinger equation. These data can produce fairly accurate results to experimental data, but rely heavily on a similarity between the subject molecule and the database molecules. Due to its restrictive scope, semi-empirical methods excel in organic chemistry calculations where relatively few elements are used with moderately sized molecules.[?] Additionally, various semi-empirical methods have been designed to produce results with close accuracies to specific properties of experimental data Two methods used in this work, AM1[?] and PM3,[?] serve particularly well at calculating data to fit heats of formation, dipole moments, ionization potentials, and structural geometries. Unlike the other methods described so far, basis sets are not used at all in the calculation of energies and properties.

1.2.2 Monte Carlo Molecular Modeling

A major problem in sampling comes with the scale of many molecules of one or more type in a single or mixed phase. A popular method of working around this problem is known as Monte Carlo methods, or MC. While not named until the 1950s, MC methods were first seen in the 18th century thought experiment Buffon's needle.[?] In his work, Buffon proposed dropping n needles of length l onto a plane with parallel lines spaced t units apart. Buffon worked out that the probability, P, of a needle crossing one of the lines to be $P = \frac{2l}{t\pi}$. Solving for π, the probability can be rearranged as $\pi = \frac{2l}{tP}$ to approximate π. Since P can also be approximated by dividing the number of needles crossing one the of the lines, h, by the n needles as $P = \frac{h}{n}$, the approximation can be expressed as $\pi = \frac{2l*n}{th}$.

This method of randomness was improved upon by Stanislaw Ulam while working at Los Alamos National Laboratory in the late 1940s by introducing markov chains to favor the probability of events occurring. Ulam shared this work with John von Neumann and together they created a program to run on the ENIAC computer capable of computing this favored version of random sampling. As the project was secretive due to being used as a part of the Manhattan Project, a collaborator named Nikolas Metropolis suggested the name Monte Carlo due to Ulam's uncle's propensity to gambling at a casino in Monaco of the same name.[?] Later dubbed Markov Chain Monte Carlo (MCMC) sampling, this allowed for random sampling to instead become a virtual statistically-appropriate sampling method. Eventually published in 1949 by Metropolis and Ulam, this laid the groundwork for modern MC methods used in modern chemical simulation packages.

1.3 Hardware

Since computation methods were developed slightly before and during the rise of modern computers, early calculations were performed by hand with minimal assistance by machines. Over time, these methods were increasingly assisted by early computers and further development eventually led to the first computational programs. These first computers, like the ENIAC and EDSAC offered computation power in the order of a few dozen to a few thousand operations per second.

For this work, the majority of calculations were computed on the Oklahoma State University Cowboy Cluster. Available since 2014, this cluster collectively offers the computing power of 3048 cores and 8576 GB of RAM, totalling 48.8 trillion FLoating Point Operations Per Second (Tera FLOPS or TFLOPS).

1.4 Software

If hardware denotes the realm of study of a computational chemist, software denotes the tools. By utilizing preexisting packages of and developing new and more advanced tools, computational chemists are able to simulate a wide variety of chemical systems.

1.4.1 Programs

While computational programs have existed for nearly 50 years, additional programs have relatively recently developed to aid in the visualization and depiction of chemical systems. Gaussian, developed by John Pople and his team is one of the earliest *ab initio* computation programs developed. Released as Gaussian 70 in 1970, it has received regular updates and capability expansions is still in use today and is one of the most widely-used computational chemistry tools available in its latest iteration, Gaussian 16. Gaussian tends to carry a lot of clout in the computational community as founder John Pople won a Nobel Prize in 1998 with Walter Kohn for his work in *ab initio* quantum mechanical systems and for being one of the oldest packages around.

In addition to Gaussian, many other packages exist with a large set of available tools for investigators. Two additional packages used in this work are GAMESS,[?] a package also in active development since the 1970s led by Mark Gordon, and NWChem,[?] a popular open source package developed by Pacific Northwest National Laboratory since the late 2000s.

Once the rounds computation has completed, investigators often report the calculated system graphically through visualization tools. These tools are also popular among any investigator wishing to represent a compound or system as more than its molecular formula. Two visualization tools used in this work are Avogadro and UCSF Chimera. Avogadro, in development since 2008, is a relatively simple molecular visualization tool designed to work across multiple operating systems across multiple languages and with many extensions.[?] UCSF Chimera, developed by the Resource for Biocomputing, Visualization, and Informatics (RBVI) at the University of California, San Francisco, focuses on more advanced representations of compounds and systems. It allows for multi-structure files to generate videos of simulations and also provides a powerful Application Program Interface for programmatically creating or altering molecules and systems.

1.4.2 Programming Languages

A final note should be made about programming languages and their usage in general and in this work. Programming languages have existed for as long as computers. From original punch cards and bitwise commands to modern interpreted languages, programming languages allows investigators to control computers to enact explicit commands. In a way, the job command files in computational tools like those in Gaussian and GAMESS are programmatically used as a programming language to tell a system to enact a calculation of type X on system Y with Z parameters. Even these tools utilize code to enact their commands, usually in older and highly efficient languages like C and Fortran. Because these tools directly interact with hardware to calculate an immense number of calculations, efficiency is key.

One language almost exclusively used in this work is Python.[?] The Python programming language has recently become one of the most used programming languages for scientific analysis. This is possibly due to Python's initial development focus of data analysis, support for extensions by the development team, and ease of use. As a scripted type languages, Python does not directly interface with hardware in assembly like the more efficient C and Fortran languages, but certain packages and extensions can take advantage of those efficiency boosts to improve Python's effectiveness. Math and science packages like NumPy[?] and SciPy[?] interface with C code to rapidly speed up complex mathematic evaluations like matrix manipulations while retaining the usability expected in Python. Additional packages like Cython[?] will take a completed Python script and compile much of it in C code to greatly improve efficiency and reduce the computational strain on the system.

As will be seen in this work, code can be used to generate and run these sets of code, effectively creating a train of code that can operate as a tool within a tool. One aspect of this is abstracting out method and basis sets to that of a computational requirement and level of accuracy. Effectively, an investigator could remove any necessary knowledge of which basis set or method is necessary for a specific system, although a tool with sufficient awareness to automatically enact this has not been published.