An Automated Script to Acquire Gas Uptake Data from Molecular Simulation of Metal Organic Frameworks

David G. van Rijswijk

Thesis submitted to the Faculty of Graduate & Postdoctoral Studies

University of Ottawa

in partial fulfillment of the requirements for the

MSc degree in the

Ottawa-Carleton Chemistry Institute

uOttawa

Candidate **Supervisor**

_____ _____

David G. van Rijswijk **Dr. Tom Woo**

© David Gregory Van Rijswijk, Ottawa, Canada, 2012

Abstract

Attention worldwide has been placed towards reducing the global carbon footprint. To this end the scientific community has been involved in improving many of the available methods of carbon capture and storage (CCS). CCS involves scrubbing flue gases of greenhouse gases and safely storing them deep underground. MOFs, a family of functionally tunable three dimensional nanoporous frameworks, have been shown to adsorb gases with great selectivity and capacity. Investigating these frameworks using computational simulations, although faster than in-lab synthetic methods, involves a tedious and meticulous input preparation process which is subject to human error. This thesis presents Dave's Occupancy Automation Package (DOAP), a software which provides a means to automatically determine the gas uptake of many three dimensional frameworks. By providing atomic coordinates for a unit simulation cell, the software acts to performs the necessary calculations to construct and execute a Grand Canonical Monte Carlo simulation, determining the gas uptake in a metal organic framework.

Additionally an analysis of different convergence assessment tests for describing the end point of the GCMC simulation is presented.

Acknowledgements

All the members of the Woo research group have my adulation for their support and contribution to an environment which cradled and honed my scientific curiosity. Firstly, I would like to thank Dr. Saman Alavi for his unending will to discuss anything of scientific merit in great and full detail. I would like to thank Dr. Chris Rowley and Dr. James Hooper for their technical guidance in the early parts of developing my project as well as Dr. Eric Des Courtis, for his guidance in programming and help in finding motivation when tasks appeared insurmountable or better yet, in providing ideas to make them conquerable. I would like to thank Andrew Sirjoosingh and Dr. Carlos Campaña for their discussions surrounding mathematics, statistical mechanics, and life in Cuba (Carlos mainly), especially those posed long after they had left the Woo research group. Hopefully, all my enquiries about the nature of their work can stop now. I would like to thank Peter Boyd for his discussions and patience in answering the same questions time and time again, while developing the *fastmc* tools used in my project. He has made me realize the benefits of posing questions through e-mail to avoid asking them three times in the same day. I would like to thank Nick Trefiak for his discussions on QSAR, the Python programming languages, and afternoon tangents into conversations about the underlying philosophy of Japanese culture. I would like to thank Mike Nohra for his discussions about “anything but chemistry”. That, more than anything, provided a much needed recourse from some of the few mundane times spent in the lab. Penultimately, and most importantly, I would like to thank Dr. Tom Woo for providing me the opportunity to pursue a graduate degree and affording me

the chance to better myself academically and personally. Although we didn't always see eye to eye, as I expect in some part is the nature of all graduate student – supervisor relationships, he is by far one of the best educators I have ever known. My experience in the graduate program, although different from my original expectation, has helped me to develop valuable skills and knowledge which otherwise I would not have been able to acquire and for that I am extremely thankful.

Lastly, I would like to thank the Lord, Jesus Christ. As promised, he has granted me the serenity to accept the things I cannot change, the courage to change the things I can, and the wisdom to know the difference.

List of Tables

List of Figures

List of Symbols

$\hat{H}$ – Hamiltonian operator

Ψ – wavefunction

E – energy

∇^2 - Laplacian operator

Z – charge of atomic nucleus

k_{bond} – bonding force constant

k_{angle} - angle force constant

D_e - bond dissociation energy

ω – bond stretching frequency

μ - reduced mass of the atoms

ϕ - dihedral angle

δ - phase factor

ε_o - permittivity of free space

q – partial charge

ε – van der Waal's well depth

σ – *van der Waal's zero energy distance*

ρ - electron density

k - Boltzmann's constant

T - temperature

Q - canonical partition function

Φ – *grand canonical partition function*

μ – *chemical potential*

p – momenta

q – position

τ - total number of collected sample points

List of Abbreviations

CCS – Carbon Capture and Storage

CPMD – Car-Parrinello Molecular Dynamics

CPU – Central Processing Unit

DFT – Density Functional Theory

DOAP – Dave's Occupancy Automation Package

ESP – Electrostatic Potential

GCMC – Grand Canonical Monte Carlo

IPCC – International Panel on Climate Change

MHz – Megahertz

MOF – Metal Organic Framework

NASA – National Aeronautics and Space Administration

OO – Object-Oriented

PBS – Portable Batch System

PSP - Pseudo-potential

REPEAT – Repeating Electrostatic Potential Extracted Atomic

RESP – Restrained Electrostatic Potential

SIESTA – Spanish Initiative for Electronic Simulations with Thousands of Atoms

UFF – Universal Force Field

VASP – Vienna Ab-initio Simulation Package

ZIF – Zeolitic Imidazolate Framework

Table of Contents

Chapter 1 - Introduction / Motivation

1.1 - Global Warming

Awareness surrounding the topic of global warming has become ever more prevalent both domestically and on an international scale in the past few decades.[1] Coming to the forefront are concerns surrounding the impact of the observed increase in the average air and ocean temperatures worldwide.[2] The associated negative implications of global warming have a large range of negative consequences to both the earth's atmosphere and as a direct result, its inhabitants. Temperature increases have been shown to affect the prevalence of hurricanes[3], increases in ocean currents[4], and glacial melting resulting in rising sea levels.[5] The threat of rising sea levels will ultimately lead to coastal flooding and other devastating effects on the earth. It is estimated that approximately 300 million people reside in the coastal area which would become flooded by a rise of just 4m, displacing its inhabitants.[6] In 2009, findings published in a study commissioned by NASA from the Goddard Institute for Space Studies noted in its recordings that the decade of 2000 – 2009 was the warmest on record. In addition, 2009 tied for the second warmest year on record.[7]

The intergovernmental panel on climate change (IPCC) reports that the increase in global average temperature is “very likely” due to the increase in concentration of greenhouse gases in the atmosphere, and more specifically so-called “anthropogenic greenhouse gases”, those resulting from the influence of human beings on nature.[8,9] Source apportionment studies have led to an unearthing of information that affirms

contributions of greenhouse gases are largely due to heavy agricultural development, industrial processing and power generation.[10,11]

The curtailing of emissions may not be a solely adequate solution to the rise in anthropogenic greenhouse gases. Speculation about the foreseeable future is widespread, but generally concedes that capture and sequestration of existing greenhouse gas is needed to offset the risk associated with climate change.[2] It is anticipated that even with aggressive curtailing the production of greenhouse gases, the atmospheric concentration of CO_2 will reach dangerous and unacceptably high levels by mid century.[2]

1.2 - Scientific Intervention

In efforts to curb the global temperature crisis, great concern has arisen in the global scientific community for the need to develop new strategies to curtail future emissions of greenhouse gases into Earth's atmosphere and to aid in the removal of existing greenhouse gases.[12]

To this end, scientists have proposed and implemented a variety of strategies involving, but not limited to, cleaner energy technologies, the development of carbon markets, and the use of carbon capture and storage (CCS) techniques. Of those carbon capture and storage techniques presently used, many rely on the use of chemical scrubbers as a means to remove greenhouse gases.[13, 14]

Improving on the efficiency of processes related to the capture of primarily carbon dioxide is considered to be key to the reduction of greenhouse gas emissions responsible for global warming.[13] Once contained, the carbon dioxide can be placed in reservoirs deep underground where it can safely be stored at high pressures in a liquid form. Due to the irreversible or energetically unfavourable profile of certain methods, capture and storage can vary in cost effectiveness and consequently becomes less feasible.[15,16]

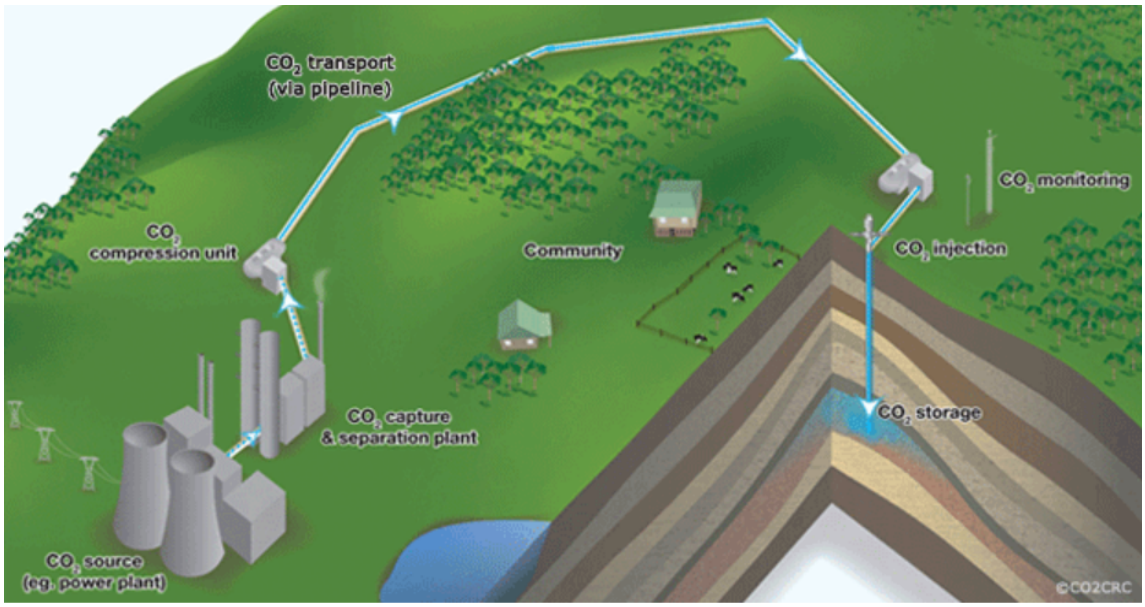


Figure 1.1 Scheme for carbon capture and storage. CO_2 gases are separated from flue gas and pumped underground to be safely stored at high pressures as a liquid. (image taken from http://www.co2crc.com.au/aboutccs/)

Recently, a new subclass of metal organic frameworks (MOFs), termed zeolitic imidazolate frameworks or “ZIF”s have been identified as being able to offer a

promising solution to the problem of effectively and efficiently capturing carbon dioxide from the atmosphere.[17]

1.3 - Using MOFs to Sequester CO_2

MOFs are nanoporous frameworks composed of metal atoms (usually Zn, Cu, and sometimes Cd) bridged to one another via a series of linker ligand molecules to form three dimensional frameworks. These frameworks can act to separate and store gases in high capacities and varying temperatures and pressures. They are easy to synthesize, highly porous, thermally stable and can be made in large quantities from low-cost materials.[17,18,19] MOFs can be structured in a large variety of ways by varying the simple building blocks they are composed of. As such, they can be designed to have a specific pore size and linker molecules can be functionalized, allowing for the creation of a large variety of tunable structures, possessing a large range of desirable properties.[20, 21]

The frameworks, containing pores and channels, allow gases to be taken up and sequestered via a physical adsorption interaction. The amount of gas that MOFs are able to uptake is largely accounted for by the dispersion interactions they encounter from the large surface area that the internal pores present. These interactions are said to be the dominant molecule molecule interaction and control physisorption to surfaces.[22]

ZIFs, a subclass of MOFs, show promise for their specificity and large capacity for carbon dioxide adsorption.[17] They are distinguished by their characteristic 145

degree angle as found between a zinc atom, an imidazole-like linker and the neighboring zinc atom. These angles resemble the O-Si-O angle found in zeolites which afford the ZIFs their qualifier “zeolitic” (see Figure 1.2 below).

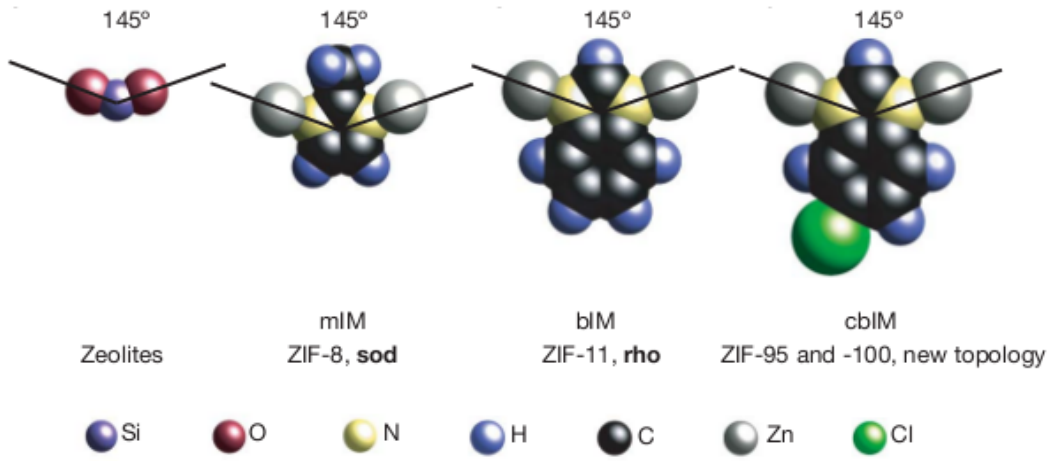


Figure 1.2 Depiction of analogous angles between zeolites O-Si-O bonds (far left) and those found in Zeolitic Imidazolate Frameworks; Zn_2(2-methylimidizolate) [mIM], Zn_2 (benzimidizolate) [bIM], and Zn_2(5-chlorobenzimidazolate) [cbIM]. Figure reproduced from ref. 19.

Enticed by their substantially high and selective CO_2 adsorption capacity, scientists have placed a great deal of attention in better understanding this subclass of compounds. The aim of exploring these compounds is to be able to identify frameworks better suited to maximally adbsorb carbon dioxide in a more selective manner.

Having compounds that are able to selectively adsorb carbon dioxide at high capacities presents an excellent tool for managing the greenhouse gas, increasing

their suitability as a tool for carbon sequestration. The foreseeable application of these types of compounds in an industrial setting is to provide a method of carbon dioxide sequestration from flue gases expelled by many industrial plants.[15] As of 2008, ZIFs have been identified which boast an uptake of 82.6 litres of carbon dioxide gas per litre of ZIF compound.[23]

Like many MOFs, ZIFs are highly porous, thermally stable, and can be regenerated for multiple uses. Although their synthesis has been challenging in the past, recent findings have shown some of them to be synthesized in a simple and straightforward manner. [23] It is hoped that reusable ZIF or ZIF-like compounds will allow us to control the emission of carbon dioxide in a more economic and energy efficient way, reducing the global carbon footprint and the associated likelihood of climate problems.

1.4 - Using Computational Chemistry to Find Better MOFs

In the search for better ZIFs, computational chemists have exploited the resourcefulness and flexibility of their discipline in attempts to model the gaseous uptake of carbon dioxide in metal organic frameworks.[16,17,24] The intent of such computational experimentation is to be able to accurately model the behaviour of these compounds with regard to their gaseous uptake and use those models with other possible frameworks to predict their potential for similar uptake. When measured upon the balance of predictive value and progress, computational methods are favoured over those of experimental alternatives because of their substantially shorter

preparation time. Although this comes at the cost of some representativeness, one might expect that the reduction in time and the flexibility provided in preparing a computer simulation far outweighs the time required to synthesize a ZIF and perform the analogous experiments to determine its gaseous uptake in a experimental lab environment.

Current processes for simulation using computational methods, although possessing faster turnaround times than traditional wet-chemistry synthesis, have not been fully optimized and require significant preparation or setup time in the creation of input files. The selection and setup of proper parameters is a cumbersome and meticulous task due to formatting restraints and manual manipulation times, and is highly repetitive in nature. Further, the simulation process (dependent on scope) may utilize different software package and utilities adding to the task's complexity as results need to be interchanged between programs.

1.5 - Project Goal

The scope of this project is to develop a software program to enhance the current simulation process for obtaining a final gas occupancy value. This is accomplished by removing the meticulous and laborious human tasks associated with the execution of simulations and assessment of intermediate results. “Dave's Occupancy Automation Package” (DOAP) essentially acts as a black box between inputted framework and final occupancy results by operating a variety of third party and in-house developed software packages. DOAP can be adequately categorized as a

“supervisory software”, one that generates, assesses and validates intermediate results for subsequent processing in the overall simulation. It acts to perform the user's task of integrating the individual software packages into the overall simulation process. Applied to the task of simulating gas uptake, DOAP dramatically speeds up useful new framework discovery by increasing the speed at which frameworks can be manipulated and assessed. The DOAP software abstracts many of the details away from the user by having the software make assessments of what needs to be performed. This allows the computational chemist to assess frameworks in a much more direct and understandable manner. The underlying complications of diagnosing situational and often functionally assessable dependencies of a successful simulation are removed from the user's concern, allowing them to place greater attention to the overall process.

Chapter 2 - Background Theory

2.1 - Chemical Theory

A variety of techniques exist for modelling a chemical system computationally. These techniques range in complexity and quality, however, the common restriction to all models is the time which they take to process. When one is computing anything, processing is measured in CPU cycles, which itself varies depending on the speed and power of the processor that the computer is operating. A processor's speed is measured in hertz or cycles/second. In a simple example, a processing unit operating at 2 MHz operates twice as fast as one operating at 1 MHz, and consequently in the same time can perform twice as many primitive calculations as the slower processor can. This being said, the limitation of the computational chemist is also the time that the model takes to process. A complex model, such as one built based on quantum mechanical theory may be able to provide a perfect answer to chemical systems but the duration of time one would endure waiting for such an answer might make it unfeasible to make any progress. Quantum mechanically based models such as full configuration interaction with infinitely sized basis sets or infinite-order perturbation theory provide such extremely detailed answers, but are said to be computationally unfeasible because of the infinite amount of computing resources they require. These techniques or techniques closely resembling these techniques are best reserved for calculations on small chemical systems or where very precise calculations of simple properties are required. In addition, the greater the number of atoms in a system that

is being modelled, the more calculations one can expect will need to be performed and consequently, the greater the time it will take to successfully process such a simulation. In order to determine aspects of solid state structure, electrostatic charges are required for the classical Monte Carlo and Molecular Mechanics calculations used in the DOAP software. In order to determine these charges, Quantum Mechanical methods must be used.

2.1.0 – Quantum Mechanics

Quantum mechanical methods lie on the "highly accurate and computationally expensive" extreme of the methods for evaluating the potential energy surface. They can be generally divided into three categories; *ab-initio* wave function methods, density functional theory methods, and semi-empirical methods. "*Ab-initio*" wave function methods, literally meaning "from scratch", seek to make use of wavefunction approximations to determine properties about the system in question. The wavefunction, denoted Ψ, is a mathematical entity representing the state of all nuclear and electronic components of the system and is said to embody all the possible information about a chemical system. The Schrödinger equation which describes the energy associated with some state of a system as it relates to the wavefunction is presented below in equation 2.1.

$$\hat{H}\Psi = E\,\Psi \qquad (2.1)$$

Where $\hat{H}$ is the Hamiltonion operator, Ψ the wavefunction of the system and E the total energy of the system.

The full non-relativistic Hamiltonian for a molecular system made up of n electrons and N nuclei is:

$$H = -\sum_{i=1}^{n+N} \frac{1}{2} m \nabla_i^2 - \sum_{I=1}^{N} \sum_{i=1}^{n} \frac{Z_I}{r_{iI}} + \sum_{i=1}^{n-1} \sum_{j=1}^{n} \frac{1}{r_{ij}} \qquad (2.2)$$

where ∇^2 is the Laplacian operator, n the number of electrons, N the number of nuclei, Z_I the charge of the Ith nuclei, r_{iI}, the distance between electron i and nulceus I, and r_{ij} the distance between electron i and j. Each *ab-initio* wavefunction method differs in its approach to evaluating this wavefunction, using different approximations of the full non-relativistic Hamiltonian for each. Examples of the methods within this category include Hartree-Fock calculations, MP2, MP4, Coupled Cluster, and Configuration Interaction calculations among others.

In addition, with the exception of the simplest system, the Schrödinger equation (equation 2.1 above) cannot be solved analytically, and so numerical evaluations require that an approximate wavefunction be generated. In all cases, a linear combination of functions is used to approximate the wavefunction. This set of functions from which the wavefunction is generated is called the basis set. Varying types of basis sets exist accounting for a wide variety of properties, in order to better model physical aspects of chemical systems such as considerations for polarizability, diffusivity, and other effects. Examples of basis sets are those such as the STO-3G, 6-311+G*, and cc-pVTZ. More accurate basis sets include larger numbers of basis functions, embodying a more complex description of the wavefunction, and requiring additional time to process. In practice we look to choose the biggest basis set which

allows us to complete the calculation in a reasonable amount of time. Further details about basis sets can be found in the book by Leach[26].

An alternative to directly solving the Schrödinger equations is to use density functional methods as described in Koch[28] and Cramer[29] which look to operate on the total electron density instead of the wavefunction of a system. These density functional theory methods are often characterized by the approximate form of the exchange correlation functional which they express. Common approximate functionals include B3LYP, PBE, BP86, and others.

Lastly, semi-empirical methods look to perform parameterized calculations which approximate the Hartree-Fock method, with the intent of speeding up the calculations involved. Common type of parameterizations are the PM3[58] and AM1[59] methods.

DFT methods are the basis for the electrostatic potential calculation that the CPMD program performs through the use of the DOAP software presented in this thesis. CPMD performs a DFT calculation on periodic systems such as those considered when looking at metal organic frameworks. The wavefunction for the system is expressed using a pseudopotential for the core electrons and plane wave basis sets to represent the valence electrons. Due to the periodic nature of the system the use of plane wave basis sets are fitting and can be manipulated in Fourier space where their computation is more easily facilitated. This accounts for some of the improved performance of density function methods over conventional methods.

In order for the CPMD program to optimize the geometry of hydrogen atoms within the DOAP software, it requires a file containing the nuclear coordinates of all atoms within the system. From these nuclear coordinates, an electron density plot is generated, which is then evaluated using a DFT functional to provide a value for the energy. Once the energy is calculated, electronic and nuclear forces can be evaluated, such that the positions of each component of the system can be perturbed and evaluated again. This process continues until the energy no longer decreases indicating a minimum energy has been found. At this point, it is expected that the geometry optimization is complete.

2.1.1 - Molecular Mechanics

While quantum mechanically related models can provide very precise and accurate answers, in order for a computational chemist to be able to achieve meaningful results in a reasonable time on large systems, they must compromise the accuracy of the system by using mathematically simpler models which are less time consuming and by comparison are said to be computationally cheap. Using such models affords the chemists the opportunity to find results which are still meaningful and predictive within a reasonable time frame. Popular models of this nature are those based on molecular mechanics as discussed in Leach[26] and Rapaport[27]. Molecular mechanical techniques do not consider nuclei and electrons independently but rather treats the atom as a whole. Each atom bears a partial electronic charge and has

spatial coordinates associated with it usually represented in Cartesian coordinates. This simplification of atomic detail is supported by the Born-Oppenheimer approximation which assumes that the fast moving electrons effectively instantly trace the trajectories of the relatively slow moving atomic nuclei. In molecular mechanics the basic information provided is used to calculate the energy of the system which is expressed as a function of five key components. These are the bond stretching potential, the bond angle potential, the bond torsional potential, electrostatic interactions, and lastly the van der Waals interaction between atoms, as described in Equation 2.1.

$$V_{tot} = \sum_{bonds} V_{bond\ stretch} + \sum_{angles} V_{bond\ angle} + \sum_{torsions} V_{torsions} + \sum_{atom\ pairs} V_{elec.} + \sum_{atom\ pairs} V_{vdW} \qquad \textbf{(2.3)}$$

Each of the individual components are expressed using relatively simple mathematical models which look to capture their energetic behaviour. These five components are generally further differentiated by saying that they are either bonded interactions (these are the stretch, angle, and torsional terms) or non-bonded interactions (the electrostatic and van der Waals interactions). Below they are described in greater detail.

The bond stretching interaction is commonly called a two-body potential and is described in its simplest form using a harmonic oscillator given by:

$$V_{bonding}(r) = \frac{k_{bond}}{2}(r - r_e)^2 \qquad \textbf{(2.4)}$$

where k_{bond} is the bonding force constant describing the strength of the bond and r_e is the equilibrium bond length, the length of the chemical bond at minimum energy.

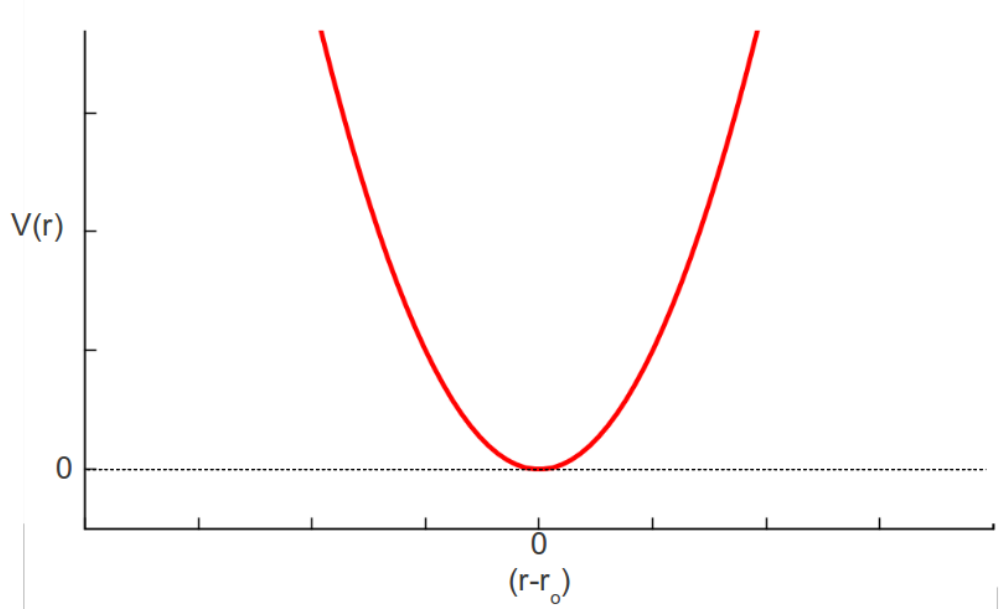


Figure 2.1 The harmonic oscillator function used to describe the bond stretching potential used in molecular mechanics. As r deviates from its equilibrium position, r_o, the angle bend contribution to the overall potential energy increases.

This function relates the square of the distance of the bond from its equilibrium length to the energy of the bond, and provides a simple yet adequately representative form from which to gauge the contribution from the bond's stretch to the overall potential energy.

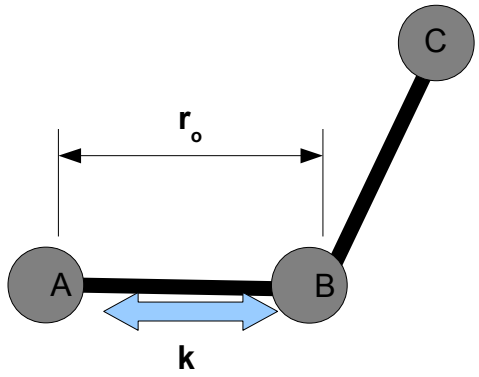


Figure 2.2 Depiction of bond stretching potential between atoms A and B, where k depicts the degree to which the bond AB resists deviation from its equilibrium distance, r_o.

Other forms of the bond stretching contribution exist and seek to describe the nature of the interaction in a more detailed fashion. One such method is by way of the Morse potential. The Morse potential differs from the harmonic oscillator in that it accounts for the dissociative nature of atoms at large separations, r. It is described mathematically as:

$$V_{bonding}(r) = D_e\{1 - e^{-\alpha(r-r_o)}\} \qquad \textbf{(2.5)}$$

Where D_e is the bond dissociation energy, r_o is the equilibrium distance of the bond, and

$$\alpha = \omega\sqrt{\frac{\mu}{2D_e}} \quad , \qquad \textbf{(2.6)}$$

where ω is the stretching frequency, and μ the reduced mass of the atoms.

As one might expect, the use of the Morse potential is more computationally expensive than the harmonic oscillator approximation due to the greater number of mathematical operations involved in each calculation. As is the general trend, finding a sufficient compromise between accuracy and speed is an important aspect of any simulation.

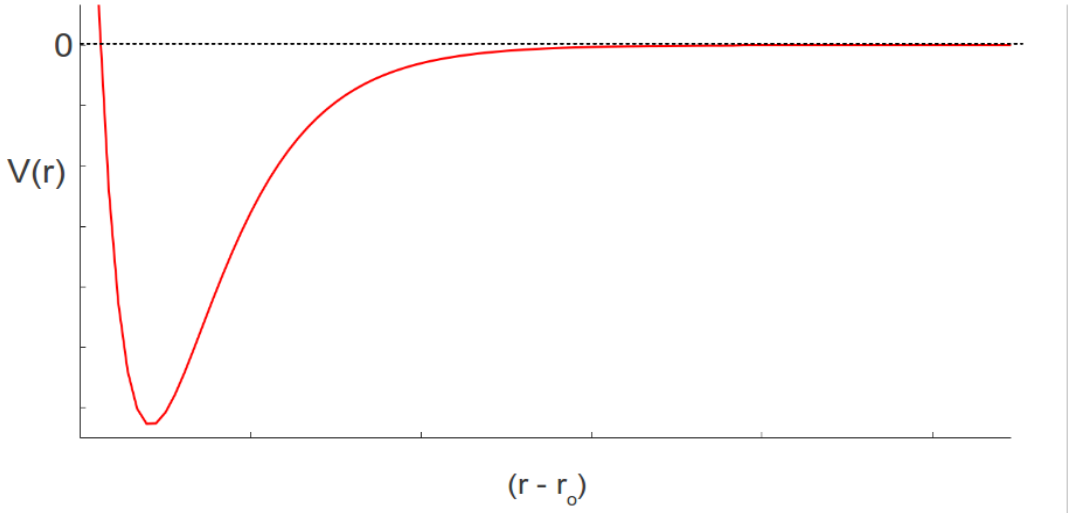


Figure 2.3 Graphical representation of Morse potential. As r increases, the resulting energy, V(r) asymptotically approaches zero energy as one would expect as a bond dissociates.

The bond angle contribution, commonly referred to as a three-body potential, describes the interaction between three atoms that can be accounted for by their angular strain.

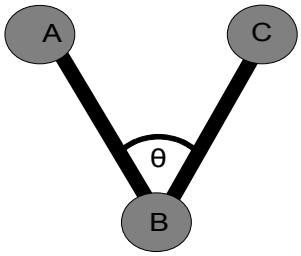


Figure 2.4 Depiction of the angular strain between bonds A, B, and C at angle θ. The factor k, from equation 2.5 below, depicts the resistance to the bending along vertex ABC.

It is generally described by a function similar in shape to equation 2.2, the two-body potential.

$$V_{angles}(\theta) = \frac{k_{angle}}{2}(\theta - \theta_o)^2 \qquad \textbf{(2.7)}$$

where θ is the angle between two successive bonds, the constant k_{angle} is a measure of the rigidity of the angle, and θ_o is the equilibrium angle between two bonds, the angle found in the minimum energy state. As the bond angle stretches away from its equilibrium bond length or compresses in a similar fashion, the potential energy of the bond increases.

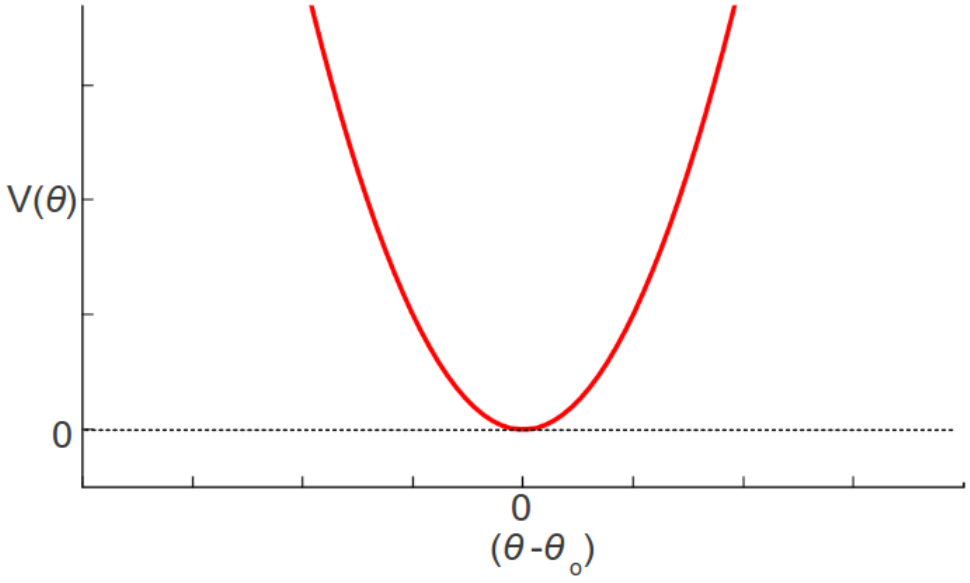


Figure 2.5 The harmonic oscillator function used to describe the angle bending potential used in molecular mechanics. As θ deviates from its equilibrium position, θ_o, the angle bend contribution to the overall potential energy increases.

The torsional contribution, commonly referred to as a four-body potential can describe both the resistance of a bond to twist upon its axis associated with sets of four atoms. The interaction is described mathematically as:

$$V_{torsion}(\phi)=\sum_{i}^{n} A_i(1+\cos(n_i\phi_i-\delta_i)) \qquad \textbf{(2.8)}$$

where ϕ is the so-called "dihedral angle" , the angle between the two planes formed by atoms A, B, C and B, C, D shown in figure 2.7 below, A is a scaling factor for the minimum energy well depth, n relates to the number of energy maxima encounter during a full (2π) rotation of the bond, and δ is a phase factor.

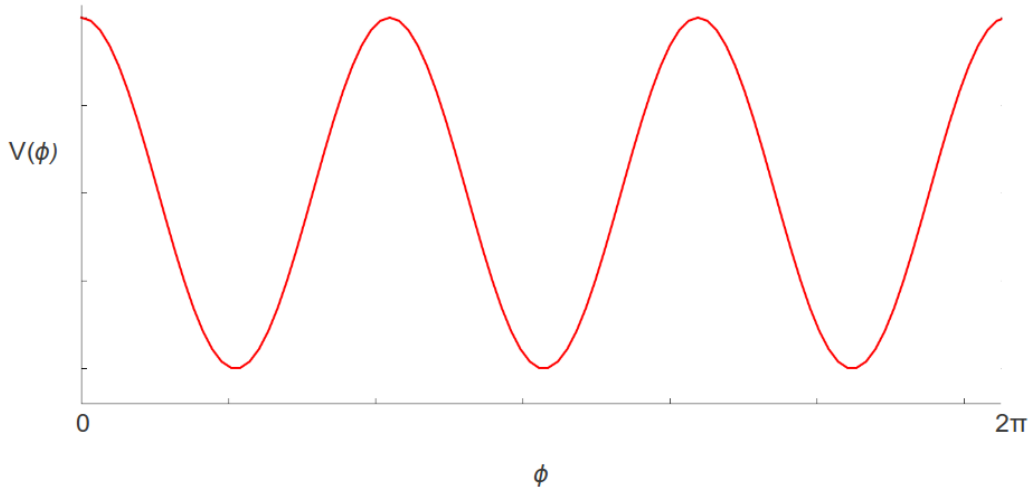


Figure 2.6 Graphical depiction of a simple torsional potential. Each trough represents a preferential torsional arrangement while each crest represents a torsional arrangement of high energy. The potential is periodic, as is the torsion about a full 2π rotation.

For clarification, a depiction of the dihedral angle is depicted in Figure 2.7 below.

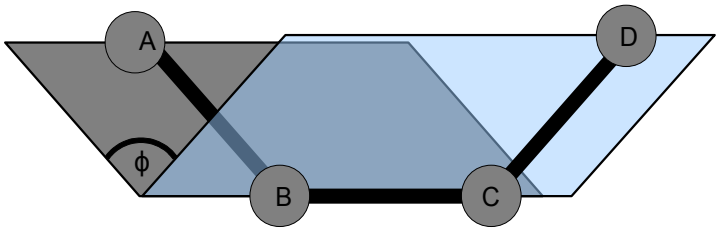


Figure 2.7 Depiction of the angle ϕ between two adjacent planes formed by atoms A, B, C and B, C, D involved in describing the torsional potential.

Unlike the bonded interactions depicted so far, the non-bonded interactions are calculated between atoms which are not directly connected to one another (termed a 1-2 relationship) or sharing a bond with a common atom (termed a 1-3 relationship). The 1-2 inter-atomic relationships are accounted for by the bond stretching potential and the 1-3 inter-atomic relationships are accounted for by the angular and torsional potentials.

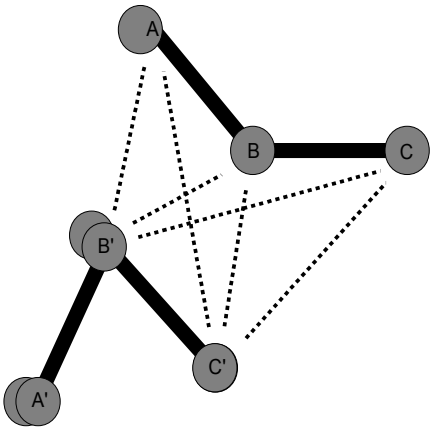


Figure 2.8 Depiction of interaction mappings between Atoms A, B, C and B', C'. Dashed lines represent interactions greater than 1-3 bonding interactions discussed above.

The non-bonded interactions are composed of the electrostatic interactions and the van der Waals interaction potentials. The electrostatic interaction is described by a Coulombic potential and is mathematically expressed as follows:

$$V_{electrostatic}(r_{ij}) = \frac{q_i q_j}{4\pi \varepsilon_o r_{ij}} \quad \textbf{(2.9)}$$

where ε_o is the permittivity of free space, q_i and q_j are the partial charges associated with atoms i and j respectively, and r_{ij} is the inter-atomic distance between atoms *i* and *j*.

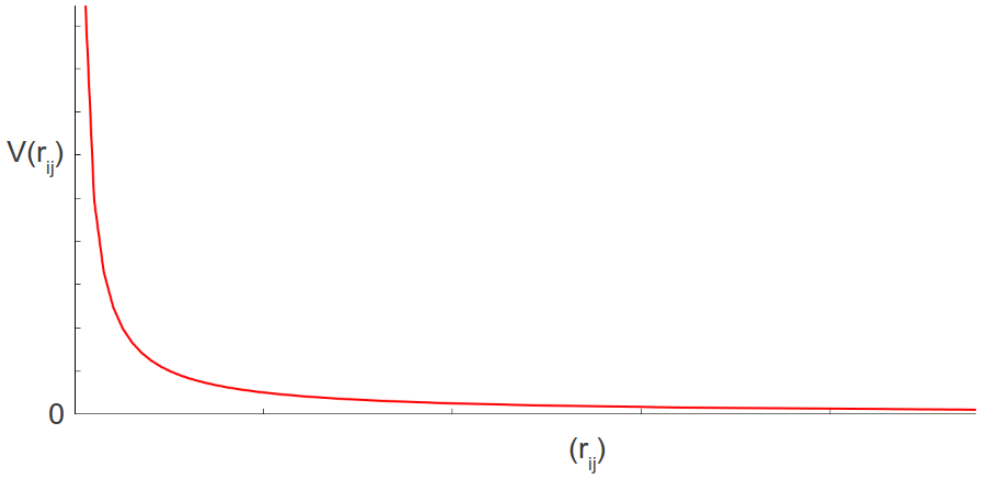


Figure 2.9 Graphical depiction of electrostatic contribution, V(r), related to the inter-atomic distance between two atoms in question,r_{ij}.

The partial charges (q_i and q_j) used in the determining the electrostatic contribution to the energy of the system are not physical observables and must be assigned. The assignment of point charges on atoms within a molecule is done using a variety of schemes, all of which attempt different methods of providing a set of charges which best represent the electrostatic potential (ESP) field which can be calculated by quantum mechanical means. It is important that the electrostatic potential is represented accurately, as unlike the other mathematical representations,

the electrostatic potential is not a bound function, and must be balanced by the van der Waals interaction.

Moreover, the electrostatic component of the potential energy has greater impact at longer ranges than any of the other forces. Special consideration must be taken when the electrostatic interaction component is evaluated. This is reflected in two ways. First, the simulation unit cell is multiplied in each dimension a set number of times. The resulting so-called simulation “super cell”, contains many more independently operating atoms than the unit cell it is composed of and allows for more chemical variability in the system to be captured as considerations of interacting components can be made at larger distances, making the overall simulation more representative of the bulk material being simulated. Additionally, periodic boundary conditions are imposed, such that atoms found at the extreme edges of the super cell interact with those on the opposite adjacent cells as if they were connected directly.

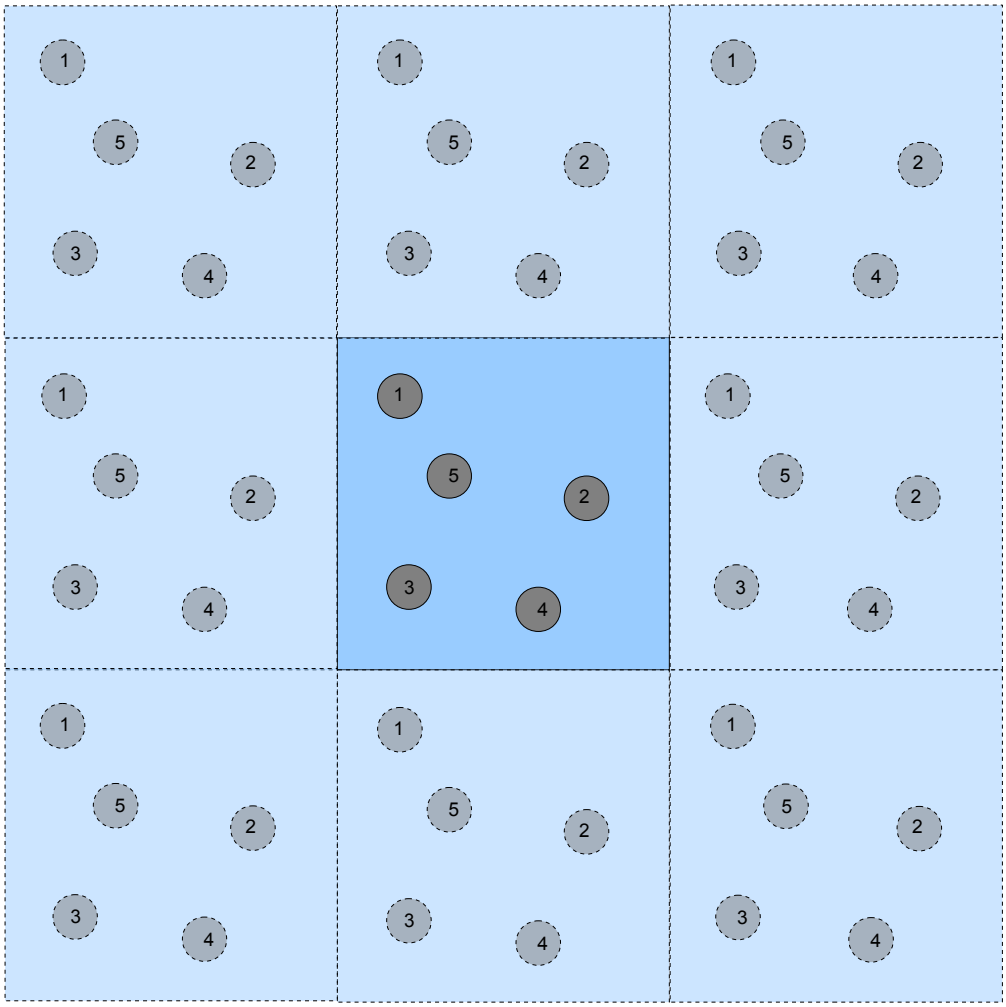


Figure 2.10 Depiction of periodic boundary conditions in two dimensions. Should atom 1 leave the cell to the left, it would re-appear entering the cell on the right side as though the unit cell (non-transparent above) were periodic in nature.

Having periodic boundary conditions presents the opportunity of having an atom interact with its periodic image. In addition, van der Waals forces which decay quickly over short distances need not be evaluated beyond a distance of 2.5σ. The interaction energy beyond this distance is said to be negligible. A cutoff value is generally implemented to prevent the needless calculation of van der Waals potential terms

beyond for atoms beyond this cutoff distance and have no contribution to the overall energy. A constraint is put in place that the cutoff cannot be greater than half the cell width. In the event that the cutoff distance exceeded half the minimum unit cell distance a self-interaction would occur.

When computing the electrostatic interaction energies in periodic systems, such as those used in this work, the Ewald summation method[29,30] is used. The Ewald summation acts to break the overall electrostatic interaction potential into two separate potentials, the short range potential and the long range potential. The short range potential is evaluated in real space where it sums quickly while the long range potential is evaluated as its Fourier space equivalent. The advantage of doing this is that the long range forces more rapidly converge in Fourier space facilitating the ease of the calculation.

Last of the potential energy components, the van der Waals contribution encompasses both the Pauli repulsion and the attractive dispersion interactions that each of the atom pairs separated more than 3 atoms apart experience. The two components are typically represented together. One common representation is the 12-6 potential or Lennard-Jones potential, described by equation 2.10 below.

$$V_{vdW}(r_{ij}) = 4\varepsilon\left[\left(\frac{\sigma}{r_{ij}}\right)^{12} - \left(\frac{\sigma}{r_{ij}}\right)^{6}\right] \qquad \textbf{(2.10)}$$

where r_{ij} is the inter-atomic distance between atoms i and j, ε is the well depth or the absolute value of the minimum energy possible, and σ is the inter-atomic distance

corresponding to an overall van der Waals contribution of zero energy. A graphical depiction of the Lennard-Jones potential is shown in figure 2.11 below.

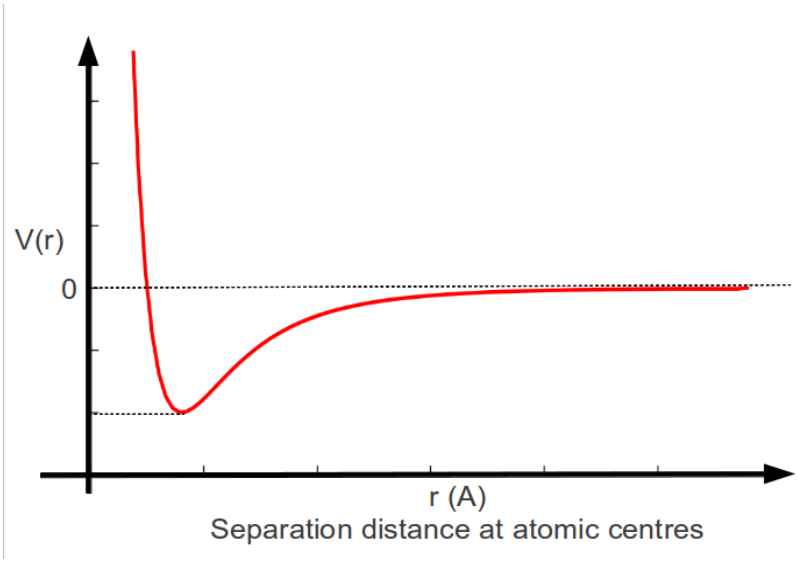


Figure 2.11 Graphical representation of the Lennard-Jones potential, showing a preferential energy well, sloped up asymptotically approaching zero energy as r increases as atoms separate infinitely and sloped up exponentially to as r approaches zero, reflecting the extremely high energy of atoms approaching one another.

2.1.2 - Force Fields

Unlike quantum mechanical models, models of a molecular mechanical nature are said to be parametrized, that is, they are developed such that the values which are used in the calculations result in answers which best match those which we see in experiment. These can include preferred geometries, relative energies, or other such criteria. Sets of parameters and the functional forms in which each component of the overall potential is expressed are termed “force fields”. These force fields are self-

consistent entities such that they do not allow for combination of one with another. Force fields can be specifically parametrized for a given family of related molecules such as amino acids or ring systems while others look to be parametrized for general use. In each case atoms are characterized not only by atomic label, but also by subtype to allow for more specifically detailed information to better represent them. For example, while in quantum mechanics one might simply specify the use of a carbon atom, in molecular mechanics, a force field may distinguish between an sp^2 carbon, an sp^3 carbon, a carbon found inside an aromatic ring, a carbon bound to a halogen, and so on. This allows the force field to capture more of the varied chemical nature of the different types of atoms potentially present in a chemical system as the theory alone, being simplistic in nature, cannot account for this variability. Specific force fields must be developed for applications where general force fields fail to properly represent the nature of the chemical system under study. In such an event, the parameters associated with the different components may be inadequate or conversely, the mathematical relationships associated with the physical behaviour of the different components of the chemical system may be inadequate to represent the system of study. Higher order contributions of entities such as multipole moment interactions, polarizing effects, and other interactions of a significant nature may not be reflected in the parameters nor accessible in the functional space mapped by the functions relating the key components to the energy.

2.1.3 - Frozen Frameworks

For the purpose of modeling gaseous uptake in MOFs the literature does not currently provide a stable force field which can account for the interactions associated with these systems. Upon attempting to simulate the MOF itself, a collapse in the geometric structure and commonly a twisting of the unit cell was observed. Upon further simulation it was found that the framework need not be permitted to move in order to provide answers which match experimental results, as the nature of the adsorption interaction of the gas molecules with the surrounding framework is of an intermolecular nature. Isothermal gaseous uptakes plotted over a range of pressures were compared between experiment and simulation and were found to be in agreement with one another, adding validation to the use of a force field which does not contain any bonded interactions; i.e. the stretching, angular and torsional contributions. The resulting framework is said to be “frozen” and solely interacts with the gas molecules present in simulation via electrostatic interactions and van der Waals interactions, so-called “non-bonded” interactions. Moreover, the simplification of the model resulting from the removal of the bonded interaction components allows for simulations containing a greater number of atoms with each simulation step taking less time to evaluate.

2.1.4 - Charge Fitting

As mentioned above, the determination of point charges arises from fitting charges to a quantum mechanically determined electrostatic potential. Point charges are not a measurable physical observable and as such cannot be compared to experiment. The electrostatic potential, a function which defines the force acting on a unit positive charge at distance r, is determinable through the use of quantum mechanical techniques. It is generally composed of two parts, an electronic contribution and a nuclear contribution. The electronic contribution is related to the electron density and is defined as:

$$\phi_{elec}(r) = -\int \frac{\rho(r')dr'}{|r-r'|} \qquad \textbf{(2.11)}$$

where ρ is the electron density of the molecule and r' the position of the electron. The nuclear component of the electrostatic potential is given by:

$$\phi_{nuclear}(r) = \sum_{I=1}^{N} \frac{Z_I}{|r-R_I|} \qquad \textbf{(2.12)}$$

where Z_I is the charge of the I th nucleus and R_I the position of the Ith nucleus.

Again, once an electrostatic potential is determined any number of approaches can be taken in order to fit an appropriate set of point charges to the potential such that it adequately represents the information of the electrostatic potential. The restrained electrostatic potential (RESP) method is one such method, however, does not account

for molecular periodicity. For the purposes of this project, the electrostatic potential is calculated using the CPMD software package.[31] The geometry of an inputted framework is fed into CPMD which then determines the electrostatic potential and stores it in an file aptly named “ELPOT”. The ELPOT file is recorded in reciprocal space, however, can be converted into real space co-ordinates using the cpmd2cube.x utility that comes with CPMD. The resulting real space based file is called a cube file. The electrostatic potential is derived from the electronic density by equation 2.9 above. CPMD approximates the wave function of the valence electrons using a plane wave basis set.[31] The remaining core electrons are described by a pseudo-potential. Having an approximation for the electronic wave function allows CPMD to perform a self-consistent field calculation to determine the electron density, ρ, through a method known as density functional theory.[32]

For the purposes of this project, once the generation of the electrostatic potential is completed, point charges are fit to it using a software implementation of the REPEAT charge fitting method developed by Campaña et al.[33]

REPEAT is a charge fitting method tailored to periodic crystalline systems. Presently, there are very few charge fitting methods which work well for periodic systems. When an electrostatic potential is calculated by a DFT package, an arbitrary reference state for the absolute energy of the system is generated. In a relative sense, the electrostatic potential is correct, that is, qualitatively, each calculation shows the same features in its electrostatic potential, however when calculations from different

packages are compared, an offset in the baseline is observed as is the case of sodalite depicted in figure 2.12 below. This becomes a problem because as a result, the point charges which are fit to the differing potentials reflect this offset in their resulting differing sets of fit atomic charges.[33]

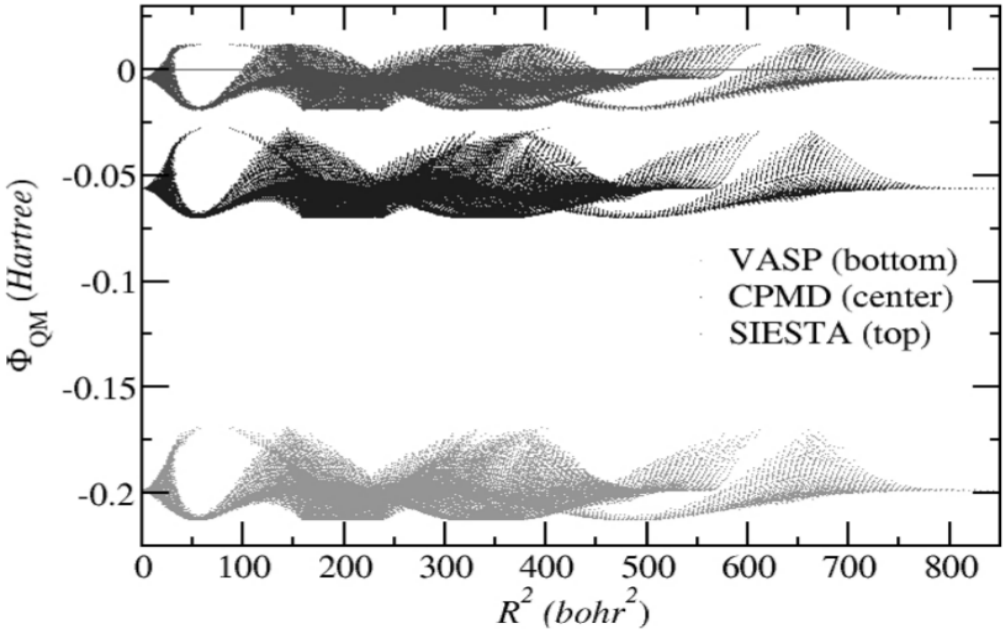


Figure 2.12 Electrostatic potential plots of sodalite from three different quantum chemical calculation packages (VASP, CPMD, SIESTA) plotted as a function of the square of the distance from the cell origin. The tabulation is performed on a cubic grid, and only points located outside the van der Waals radii centered on each atom are shown. Reproduced from Campaña, 2009.[33]

Within each package, this problem arises from the use of an error functional which fits charges based on absolute values of the potential. The REPEAT method

accounts for this problem by introducing a new error functional. The key to this new functional is that it acts on the relative difference of the potential.

2.1.5 - Statistical Mechanics

Statistical mechanics provides a means by which macroscopic observables can be determined through the calculation of microscopic properties.[34,35] An understanding of how microscopic information relates to macroscopic observables is accomplished through exploration of the notion of the statistical ensemble. Ensembles are grouped "images" of an initial system constructed for use in a theoretical frame of analysis.

Each image or copy system operates independently, but wholy the entire ensemble represents an expected distribution of states among a bulk extension of the unit system. Ensembles are characterized by the restrictions imposed on the systems they contain. The basic ensemble, where the system is kept under constant volume, with a constant number of particles, and at a constant temperature, is called the canonical ensemble. Theoretical assertions can be constructed about the nature of the ensemble given its constraints. For instance, the energy distribution for different quantum states of a canonical ensemble can be expressed using the relationship:

$$P_i = \frac{e^{-\beta E_i}}{\sum_i e^{-\beta E_i}} = \frac{e^{-\beta E_i}}{Q} \qquad \textbf{(2.13)}$$

where $\beta = \frac{1}{kT}$, E_i denotes the i^{th} energy level, P_i is the probability of finding the system in an energy level, E_i, k is Boltzmann constant, T the temperature, and Q the canonical partition function.

The canonical ensemble is of little use in the modelling of gaseous uptake. The Monte Carlo methods as presented in Frenkel and Smit[60], which will be discussed in section 2.1.6 below, rely on a simulation whereby the temperature is fixed, the volume is fixed, and the number of particles inside the system varies as to account for the proper number of adsorbed gas molecules. These constraints resemble the characteristics of the grand canonical ensemble with the added restriction that the chemical potential of the system, μ, remains the same. From the analysis of such a theoretical ensemble we will be able to express the properties of the system under simulation, but also gain an appreciation for the restrictions of the sampling procedure that must be imposed in order to achieve such a representative and therefore macroscopically relatable ensemble. The probability distribution of energy states for the grand canonical ensemble is equivalently describable and follows the relation:

$$P_{N,i} = \frac{e^{-\beta(E_{N,i} - N\mu)}}{\sum_N \sum_i e^{-\beta(E_{N,i} - N\mu)}} = \frac{e^{-\beta(E_{N,i} - N\mu)}}{\Phi} \qquad \textbf{(2.14)}$$

where $\beta = \frac{1}{kT}$, $E_{i,N}$ denotes the i^{th} energy level of N particles, $P_{N,i}$ is the probability of finding the system in an energy level, E_i with N particles, μ *is the chemical potential*

associated with one particle, k is the Boltzmann constant, *T* the temperature, and *Φ* the grand canonical partition function where,

$$\Phi(\mu, V, T) = \sum_{N} \sum_{j} \exp(N\mu / kT) \exp(-U_{Nj} / kT) \qquad \textbf{(2.15)}$$

While for molecular mechanics a classical counterpart to the probability function is needed, the probability is still directly proportional to $e^{-\beta(E_{i,N} + N_{\mu})}$ and need not be explicitly calculated for its use in the Monte Carlo method.

2.1.6 - Monte Carlo and Metropolis Monte Carlo Methods

Ensemble averages can be used to calculate time averages. This approximation is known as the ergodic principal. The ergodic principle states that over long periods of time, the distribution of states that a system will obtain will approach the instantaneous distribution of states in an extended system of the same composition. In general, the expectation value of some property, A, can be defined using an ensemble where formally:

$$\langle A \rangle = \iint dp^N dq^N A(p^N, q^N) \rho(p^N, q^N) \qquad \textbf{(2.16)}$$

where *p* and *q* are the momenta and positions of the system's particles respectively, and the term $\rho(p^N, q^N)$ defines the probability density of states with momenta p^N and

positions q^N over all N atoms. In the case of the grand canonical ensemble this probability density is defined as $P_{N,i}$ where:

$$P_{N,i} \propto e^{-\beta(E_{i,N}+N_{\mu})} \quad \textbf{(2.17)}$$

The remaining component Φ can be viewed as a normalization factor. Calculating the value of Φ is infeasible and consequently, calculating the probability density, $\rho(p^N, q^N)$ is likewise infeasible. Metropolis *et al.* have circumvented this mathematical inefficiency by proposing the Metropolis Monte Carlo method which extends the Monte Carlo method and removes the dependence of the probability density function on the grand canonical partition function, Φ.

Instead of addressing the problem, Metropolis was able to propose an approximation of the expectation value of property A, with the restraint that the sampling be performed in a way which provides a probability distribution that approaches the limit of the grand canonically distributed probability.[36] This method is referred to as the Grand Canonical Monte Carlo (GCMC) scheme. The approximation is defined as:

$$\langle A \rangle_{observed} \approx \frac{1}{\tau} \sum_{i=1}^{\tau} N(p_i, q_i) = \langle A \rangle_{simulation} \quad \textbf{(2.18)}$$

where τ is the total number of collected sample points and

$\lim \langle A_{simulation} \rangle_{\tau \to \infty} = \langle A_{observed} \rangle$, since $P_{GCMC} \to P_{grand\ canonical}$ as $\tau \to \infty$.

The momenta dependent temperature terms are separable from the positions and can be accounted for by a set of momenta following the Maxwell distribution. For the grand canonical ensemble, with constant temperature, the property expectation can

then be expressed in terms of the positions only. This is the basis for the Grand Canonical Monte Carlo simulations that will be used to determine the gaseous uptake value of the frameworks. The adequate method of sampling for a Grand Canonical Monte Carlo simulation is achieved by the following proposed algorithm whereby a frozen framework including particles of gas are operated on by various “moves”:

- A move is randomly performed on some particle in the system:
  - A particle is displaced from position s to some new position r
  - A particle is deleted
  - A particle is created in some random position
- The energy of the configuration is calculated using the molecular mechanical force field.
- If the new energy of the system is lower, the move is accepted
- If the new energy of the system is higher, the move is accepted with some probability, P

The probability P, or acceptance criteria provides a value between 0 and 1. Should a random value generated in parallel between 0 and 1 be greater than the value of the acceptance probability then the move is rejected. Should the random value be lower that the value of the acceptance probability then the move is accepted. The acceptance criteria for each move in the Grand Canonical Monte Carlo are defined as:

$$Acc_{Displacement(r \to s)} = min\left\{1, e^{-\beta(E_s - E_r)}\right\} \quad \textbf{(2.19)}$$

$$Acc_{Insertion(N \to N+1)} = min\left\{1, \frac{\beta VP}{N+1} e^{-\beta(E_{N+1} - E_N)}\right\} \quad \textbf{(2.20)}$$

$$Acc_{Deletion(N+1 \to N)} = min\left\{1, \frac{N}{\beta VP} e^{-\beta(E_N - E_{N+1})}\right\} \quad \textbf{(2.21)}$$

Common to all three of these acceptance criteria are the contribution of the temperature, encapsulated in the β term. At higher temperatures the value of the Acceptance criteria increases, affording greater probability that the move will be accepted. This allows for an increased probability of the sampling of higher energy states as would be expected when the temperature is high. Additionally, this expression ensures that larger changes in energy are less likely to be accepted than smaller changes in energy at constant temperatures.

In the DOAP software, a program called *fastmc* written by Peter Boyd is used to accomplish the Grand Canonical Monte Carlo simulation.

2.2 - Computational Theory

2.2.1 - Programming

The entirety of this project is virtual. It is effectively a vast set of complex instructions carried out by computer systems. While the instructions are very complex,

computers themselves are very simple in nature. They are able to perform only simple operations but do so at extremely fast rates. The central processing unit of a computer interprets and processes information represented in binary format. Having a user interpret and convey information to the processing unit directly is highly unfeasible, because of the tremendous amount of work and detail involved in encapsulating information in what would be tremendously large binary inputs and interpreting tremendously large binary results. In addition, it is incredibly inefficient, as the computer can resolve binary arguments many many orders of magnitude faster than a user can create them. In order to harness the power of the computer fully and make effective use of the computer, the machine-level detail of the information must be removed from the user's perspective. This is done through the use of layers upon layers of abstraction whereby simple functions are composed to create higher-level functions which in turn are composed to make even higher level functions and so on. These high level function can then be called upon and evaluate the functions which compose them. Programming languages are a type of abstraction. They are a means by which to assert information between the needs of a user to input and receive intelligible information to and from the computer and the needs of a computer to receive, process, and output bit-like information from the user. Essentially, a programming language acts as an interface between the high level world of the user and the low-level world of the processing environment. There are many paths by which to abstract information from the point of low-level to high-level functionality and as such, different types of programming exist. Each one is characterized by a different style of composition, organization, and execution and as such, entails its own benefits,

and drawbacks. Programming languages are generally written with a type of programming in mind, however some so-called hybrid languages seek to accommodate combinations of styles in order to gain from the benefits of multiple paradigms and mask the deficient characteristics of a language by circumventing them with alternative approaches from others.

2.2.2 - Structured Programming

One of the common styles of programming is called structured programming. In structured programming instruction clauses are separated into distinct sets including block structures, subroutines and control of flow structures which may overlap or nest procedural information inside one another. Generally, program code written in this way resembles the structure of natural spoken languages and are easily related to and conveyed by novice programmers who find this type of programming very approachable. Structured programming enables but offers little enforcement of organization and code is unrestrained from becoming convoluted. The structured programming theorem outlines three components of the paradigm; sequence, selection, and repetition.[37] Sequence refers to code of a procedural and ordered nature, selection refers to variable execution of code based on the evaluation of some statement, and repetition refers to the evaluation of code on a repetitive basis until some condition indicated in the global or local state of the code is achieved. The structured programming theorem promotes these three elements as a sufficient means by which to express any computable function.[38]

2.2.3 - Object-Oriented Programming

Object-oriented (OO) programming is a paradigm of programming which looks to encapsulate code into distinct constructs called “objects”. Objects contain both information and functions and can be composed of other objects. Additionally, they can borrow functions (often called “methods” in OO) from one another and conversely extend the functionality of other objects. Object oriented programming is well received because it provides the user with the necessary structure for organizing and classifying the different components of their overall program by modelling them against the “real world”. This compartmentalization allows objects to be transferred from one application to another, but couples elements of a program together, which as will be discussed is often not a favourable property. The term object-oriented was first coined by Alan Kay in the late 1950s when at its inception it was defined by six rules outlining the paradigm.[39] They are:

1. Everything is an object.
2. Objects communicate by sending and receiving messages (in terms of objects).
3. Objects have their own memory (in terms of objects).
4. Every object is an instance of a class (which must be an object).
5. The class holds the shared behaviour for its instances (in the form of objects in a program list)
6. To evaluate a program list, control is passed to the first object and the remainder is treated as its message.

As can be seen by rule six, by definition an object is coupled to all other objects, directly or indirectly.

2.2.4 - Functional Programming

Stemming out of an era before the dawn of electronic computing machines, functional programming has its root in lambda calculus, and was originally proposed by

Alanso Church in the 1930s.[40] Functional programming can be construed as highly mathematical in nature, but provides a very simple and powerful means of expressing very complex instruction sets.[41] Many tens of lines of structured programming can often be expressed in a few lines of functional programming code.[42] When expressed correctly, functional programming can be written in such a fashion that is highly expressive in both an instructional context as well as a user interpretability context. Because of their mathematical nature, functional programs are formally provable. Each function in a functional program differs from those we see in programs based on imperative programming paradigms such as object-oriented programming and structured programming, because the functions themselves are not able to augment the global state of the program. Essentially, functional programming has no side effects because the program has no global state.[43] Functions of this nature are termed “pure” functions. They are entirely decoupled from the rest of the program such that, if their result was not used, the code could be removed from the program and not affect other functions. Likewise, if a pure function is called multiple times with the same inputs, it returns the same output each time.

2.2.5 - Nature of Python

The entirety of the DOAP software is written in Python. Python is a “high-level” hybrid programming language. It supports the use of functional and object-oriented programming as well as imperative styles such as structured programming.[47, 48] Python is an interpreted language, meaning that the program code is not compiled before execution, but rather is processed through an interpreter program that performs the instructions written in the code as the program runs. The core philosophy of Python

emphasizes code readability. It was developed in such a way that the code can be written with great expressive quality and is very easy to use. These strengths of the Python programming language have been embodied in the development of the DOAP software.

2.2.6 - Functions

Groupings of code which need to be executed more than once can be declared as distinct named segments of instructions called functions. By doing this, the instructions to be carried out need not be reiterated in full within the main branch of the code each time they are required. References to the function by name are instead written, with a list of requisite parameters attached in some form to the name. The parameters are said to be “passed” to the functions. By structuring the code in this manner, the code becomes simpler and much easier to understand as the function name helps to simplify understanding the intent of the segment of code. The user need not worry about how the task is performed in the code of the function, merely that it is. In the event that the function does not return a correct value, the code associated with the function can be replaced or updated without having to change the code which uses the function. When a function is called, the parameters are passed into it are operated on within the context of the function. The environment of the function is termed the local scope. Parameters can be passed into a function by value or by reference. Passing the value, creates a copy of the information to be manipulated by the function. Any change to the value inside the function is not reflected in the original

occurrence of the value from which the copy was made. When information that is passed is of a structure which has greater depth to it, generally it becomes inefficient to generate a copy of such a large construct. In this case, many languages choose to have the calling program pass a reference to the parameters being passed. Instead of a copy of the value, the location in the computer's memory is sent to the function receiving the parameter. It is important to make this distinction, because any changes made to the information at that location, is reflected in the calling program's view of this information, as it references the same memory location.

2.2.7 - Side effects

Any time a function is able to act on information outside of its own scope, an opportunity is presented for code to have side effects. Side effects are any unintended consequence of the execution of a piece of code. A function should be limited to performing a single simple task. Functions that are able to act on the global state of a program can become unruly, and complicate things needlessly. It is therefore preferential to structure a program such that a function operates on only copies of the information which it is passed, and provides an answer which the calling program can ascertain the validity or proper execution of and manage its resulting process from there.

2.2.8 - Decoupling of states / modularity

Modularity is a practice by which code is segmented into distinct groups that do not rely on one another.[49] A function that relies on the use of information tied to another function is said to be coupled to that function. Changing the value of data in one function reflects in a change of how the other function behaves. Segmenting the program into modules not only helps to organize the program into useful libraries of instructions, but decouples the components of the program which prevents the occurrence of side effects, previously discussed. The intent of programming is to abstract functionality, not to compose lists of automated sequential instructions. Ideally-decoupled code looks to perform a simple task which is a composition of other simpler tasks. Breaking a program down in this way facilitates the greatest amount of code reuse, which in turn makes coding easier to accomplish, and also makes comprehending code much easier, as each segment of code only has one focus and is only composed of a few slightly lower-level functions. That being said, in some cases, the coupling of code is inevitable.

2.2.9 - Unit Testing

Testing of code is a highly important component of the software development cycle. Errors in code can lead to unstable states of the program or worse yet, stable states which report erroneous answers and results. While syntactical errors are easily identified upon compilation and fixed, problems in the logic of the instructions, called bugs, are not as easy to identify. The process of removing bugs can become quite

complicated if time has not been properly invested in making code manageable. Having code that is simple in context and that explicitly declares its intent allows the user to identify problems quickly. Further, having decoupled code ensures that the offending unit of code can be replaced and have minimal influence on the rest of the code. Too often does decoding a bug result in following a series of coupled functions and painstakingly tracing their interactions with one another to resolve the issue. This practice is avoided by making clean, clear, and effective code.

Having the code in decoupled units of instruction allows for the application of unit testing. Unit testing provides a framework by which code can be validated, analyzed and restructured.[50] The goal of unit testing is to set up a framework by which all the components of a program can have their logic tested in an automated fashion and provide a rigor or standard for which any rewritten function can be said to accomplish at least as much as the defunct code it is replacing. In unit testing, simple test functions are generated with the intent of "breaking" the functions that the program is composed of. A test consists of an evaluation of a function, followed by an assertion about the nature of the result. Assertions vary in nature and can themselves be results of functions. It is important that the functions themselves are as simple as possible so that the unit tests are equally simple. Having many components to a function increases the complexity of the unit test required in order to assess each case. Generally the unit tests of a function look to exploit the function by presenting the "edge cases" of the parameters involved.[50]

A clear example is given by a function that performs simple division:

```
dividing_function --> (numerator_parameter, denominator_parameter):
        result = numerator_parameter/denominator_parameter ;
        return result
```

In the above example, a function which takes two inputs, numerator_parameter and denominator_parameter, uses the values of those two variables to evaluate and return the quotient of that operation. An obvious edge case involved in a unit test might entail sending a denominator of the value zero.

Clearly, should a zero be sent, the function would return a “divide by zero” error and fail, given that infinite is not a regular countable number. This would indicate that the function does not suitably handle this case, and the function would be recoded.

A new function:

```
dividing_function --> (numerator_parameter, denominator_parameter):
        if denominator_parameter = 0 then return INFINITY;
        else: result = numerator_parameter/denominator_parameter ;
        return result;
```

would pass the same unit test. The intent of the unit test is not to bombard the function with non-interpretable parameters, but rather to test that the logic of the function works using valid inputs.

For example, while passing the denominator “Q” to the division function would break the function, it is not a valid test case as the function isn't intended to process characters. In reality there are a series of test cases for each function. By constructing test cases around the function, the limits of the function are characterized, and suitably the intent of the function is conveyed in an indirect manner.

Should a new function with the same inputs be needed to be coded from scratch, it is expected that it should minimally pass the tests already present should it look to embody the functionality of the previous incarnation of the function that it is replacing. Unit tests that verify a common function or group of functions are grouped together in test suites. Test suites can also include other test suites. This affords the programmer a method by which to outline different series of tests, should individual components of the code be prone to more failure than others.

2.2.9 - Test-driven development

The concept of unit testing is further extended into a method of software development called test-driven development. In test-driven development the behaviour of a function is first outlined by edge cases represented in the test case for a function. [51] Only after which is a function written to pass these tests. Should a function fail in the execution of the code, the failure is examined and a test case is written to reproduce the source of failure. The function is then recoded to account for this case and the new test case remains in the unit test suite for that function in order to ensure that any future incarnations of that function are equally able to handle that case.

Unit testing may also be used to identify components of a system that are not compatible with another system. Should the code depend on some fixed environmental constant of a system, such as the location of a file, or the presence of a piece of hardware, unit testing can provide an easy means of identifying that the component is missing as the entire test suite or individual tests constructed for the functions associated with this component will fail, bringing to the programmer's attention that a new module may need to be coded in order to interact with this analogous component of the new system.

2.2.10 - Practices used in developing DOAP

The DOAP software was written with the intention of providing a clear, concise, highly understandable, adaptable code that can be easily approached. The libraries that have been written to support this code offer a modular and extendable framework which can be used to abstract complex procedures involved in molecular mechanical simulation and are not limited to use only by this project. The core program, acting as a overseeing program for other simulation programs, is not the rate limiting step of the simulation process. The project was written in Python 2.4. Python is considered a high-level language, and to this effect, the speed at which the code executes is not a primary concern. The use of Python for this project is rationalized in that having ultimately efficient code was a secondary concern to the flexibility and expressive

quality of writing that Python offers. These advantages have been exploited considerably in the naming conventions applied and simplicity of the code. Variable and function names are written explicitly such that there is no ambiguity in their intent. This practice, called “self-documenting code”[52], has been rigorously upheld during the development of DOAP. Functions have been kept to a minimum line count and have been broken down to perform as least of a task as possible. Where possible, complex nested selection structures can been replaced with simple and powerful list comprehensions. Side effects have been minimized by attempting to decouple the states of as much of the program as possible. Where decoupling was not inherent to the organization of the structure, objects were used to facilitate comprehension and operation. Classes of objects used the “composition” design pattern over the “inheritance” design pattern as suggested in the Gang of Four book on design structures.[53] All user data are validated prior to program execution. This is done by use of a verification module and syntax checking of components. The validating functions verify the nature of the values associated with certain simulation environment variables by matching them to accepted regular expressions. Lastly, full unit test coverage is provided to ensure the integrity of the code and provide a means of encapsulating the intent of all functions within the code base.

Chapter 3 - Computational overview

This chapter provides a high level overview of the nature of the operation of the DOAP software. Below is a terse and technical description of the rudimentary aspects of the system that the user should be aware of in order to operate it effectively. Further detail, although not required to operate the system, will be provided in Chapter 4 for those wishing to extend or implement features of the code in other softwares.

3.1 - Structure of package

Determining the uptake of a guest gas in a metal organic framework requires the user to provide the co-ordinates of the frameworks unit cell along with its unit cell vectors. It is important to note that the eventual simulation that will provide the gas uptake value is done with a frozen framework, and so intramolecular concerns, such as bond lengths, angles, and torsions, do not need to be considered when building a force field for simulation.

The general scheme for performing the overall simulation is laid out below:

- The user provides the program input files. They are:
    - Coordinates of the unit cell atoms in xyz file format.
    - Information regarding the construction of the guest molecule.
        - Relative xyz coordinates.
        - Charges for all atoms of the guest.

    - A set of Lennard-Jones van der Waals parameters for all atoms in the simulation.
    - An input file containing all the parameters associated with the simulation. (Details are provided in section 3.4)
- The program validates the construction and integrity of the input files, verifying that all requisite information is present and in proper order. This includes:
    - Verification of minimum cut-off distances. This is important to avoid self-interactions when the electrostatic potentials are evaluated.
    - Verification that files and paths exist.
    - Verification that the values of input parameters are reasonable by matching them against predetermined accepted ranges and possible values using regular expressions.

- The program constructs a CPMD input file and submits it to CPMD to generate the ESP and optionally optimize the geometry of any hydrogen atoms.
- The program monitors CPMD for completion.
- Once completed, the CPMD output files are verified for completion.
- A cube file is generated from the CPMD ELPOT file using cpmd2cube.x.
- The cube file is inputted into REPEAT to develop point charges for the atoms.
- The resulting charges are parsed out of the REPEAT output and fit to the unit cell coordinates.
- The program constructs a super cell out of the unit cell coordinates.

- The program, generates CONFIG, CONTROL, and FIELD files, the inputs for the *fastmc* GCMC package.
- The program submits the *fastmc* run and monitors it for completion.

3.2 - File Structure and Intent:

The DOAP software is broken up into 20 components. They are 2 DOAP executable files from which the user runs the software, 3 libraries of code specific to DOAP related tasks, 7 general libraries of code, 3 data modules, 4 input files for the DOAP system and 1 file required for the use of Python features.

DOAP executable files(2):

get_gas_uptake.py - main executable file for stand alone operation.

submit_gas_uptake_job - script to submit DOAP job through PBS queueing system.

DOAP support libraries (3):

data_validation.py - library used to validate inputs for all stages of simulation.

doap_logging.py - logger for DOAP framework.

doap_tools.py - parameter management tools and other functions used in DOAP.

General libraries (7):

cpmd_tools.py - library used to build CPMD inputs, select PSP files, *etc.*

fastmc_tools.py - library used to import and write all *fastmc* simulation input files.

gauss_tools.py - library used to make Gaussview viewable files.

molecular_mechanics_tools.py - library used to adjust charges and work with vdW files.

queue_tools.py - library containing objects for each type of simulation job.

repeat_tools.py - library used to build repeat files, parse charges from output, and retrieve completed jobs.

xyz_tools.py - library to handle all mathematical manipulations of coordinates.

Data modules (3):

chem_constants.py - module storing physical chemical constants.

exception_classes.py - module storing all custom exception classes used.

validation_criteria.py - module storing all regular expressions and variable types for inputs.

Input files(4):

DOAP_OPTIONS - main parameter file, stores all user inputted values, locations.

INPUT_MOF_NAME - XYZ formatted input file. Stores input coordinates.

GUEST - guest information file. Stores, parameters, coordinates.

vanderWaal_parameters - set of Lennard-Jones sigma and epsilon values for each atom type in the simulation.

Other files(1):

__init__.py - required for certain Python functionality.

3.3 - Functionality and Limitations

The DOAP system is centered around the JOB_OPTIONS file which outlines all the directives involved in performing the overall simulation. Below is a detailed list of all directives involved in the DOAP system, followed by a brief description of each, outlining their purpose.

MAIN_CHARGE_PACKAGE [PACKAGE NAME]**:**

The MAIN_CHARGE_PACKAGE directive selects for the software that will be used to calculate the electrostatic potential. Although the only current available option is to use CPMD, the package has been coded with this flexibility to allow for future additions of other packages such as VASP or SIESTA.

MAIN_POINT_CHARGE_PACKAGE [PACKAGE NAME]**:**

The MAIN_POINT_CHARGE_PACKAGE directive selects for the software that will be used to fit point charges to the ESP grid calculated for the framework. Although the only current available option is to use REPEAT, the package has been coded with this flexibility in mind to allow for future additions of packages related to point charge fitting.

MAIN_GCMC_PACKAGE [PACKAGE NAME]**:**

The MAIN_GCMC_PACKAGE directive selects for the software that will be used to perform the final gaseous uptake GCMC simulation. Earlier revisions of the package maintained support for both the DL_POLY GCMC code written by Andrew Sirjoosingh

and the *fastmc* code written by Peter Boyd, however, due to the deprecated state and lack of support for the DL_POLY GCMC code, the option to use the code has been removed.

GENERAL_VDW_FILE [FILE NAME]**:**

The GENERAL_VDW_FILE directive defines the location of a file storing the van der Waals potential Lennard-Jones parameters, epsilon and sigma. Currently the universal force field (UFF) parameters described by Rappé *et al.* [51] is included with the package and is selected by default in the JOB_OPTIONS file.

GENERAL_SYSTEM_CHARGE [CHARGE]**:**

The GENERAL_SYSTEM_CHARGE directive describes the charge of the system. It is used multiple times, both in the charge fitting routine and the ESP calculation, although it will remain zero for the case of all gaseous uptake simulations given the periodic nature of the systems being simulated. Should this variable obtain a non-zero value, the net charge would become infinite and cause the system to halt.

GENERAL_SYSTEM_TEMP [TEMPERATURE]**:**

The GENERAL_SYSTEM_TEMP directive describes the temperature of the system. This is used mainly in the construction of the *fastmc* simulation input files, but remains globally available with the GENERAL_ prefix to be flexible for future applications.

CPMD_NUM_PROCESSORS [N]**:**

The CPMD_NUM_PROCESSORS directive is used when submitting the CPMD ESP calculation to the cluster. It conveys the number of computer processors, [N], that the calculation will be performed on.

CPMD_CONVERGENCE_CRIT_ORBITALS [CRITERIA]**:**

The CPMD_CONVERGENCE_CRIT_ORBITALS directive is used when constructing the CPMD ESP calculation input files and directs the criteria by which the optionally selected geometry optimization is considered complete.

CPMD_MAX_ITERATIONS [N]**:**

The CPMD_MAX_ITERATIONS directive is used when constructing the CPMD ESP calculation input files and directs the maximum number of iterations that the geometry optimization will perform in converging.

CPMD_PSP_PATH [PATH]:

The CPMD_PSP_PATH directives declares a path location to a directory containing the pseudo-potential files that the user expects to use for the CPMD ESP calculation.

CPMD_FUNCTIONAL [FUNCTIONAL]**:**

The CPMD_FUNCTIONAL directive declares the default type of functional associated with the pseudo-potential file which the CPMD system will look to use in the ESP calculation.

CPMD_PSP_TYPE [PSP_TYPE]**:**

The CPMD_PSP_TYPE directive declares the default type of PSP that will be associated with the pseudo-potential file which the CPMD system will look to use in the ESP calculation.

CPMD_LMAX_FLAG [L]**:**

The CPMD_LMAX_FLAG directive declares the default value assigned in the construction of the CPMD input files relating to the non-locality of the pseudo-potential, declaring the maximum l-quantum number, [L].

CPMD_CUTOFF [X]**:**

The CPMD_CUTOFF directive provides the distance for the plane wave energy cutoff in Angstroms for the CPMD ESP calculation. It is used in the construction of the input files for CPMD. The size of the cutoff is directly related to the size of the basis set as trying to represent the wave function at a greater distance requires more basis functions.

CPMD_OPTIMIZE_HYDROGENS [Y/N]**:**

The CPMD_OPTIMIZE_HYDROGENS directive, when set to true, instructs the DOAP system to attempt to generate input files for the CPMD ESP calculation which will first perform a geometry optimization on the hydrogen atoms of the system. Hydrogen atoms are commonly misaligned in crystallographic data. Should there be no hydrogen atoms present in the unit cell and hydrogen optimization is selected for, the DOAP system will avoid constructing the CPMD input files with instructions to optimize hydrogen atoms.

REPEAT_PERIODICITY [0/1]**:**

The REPEAT_PERIODICITY directive instructs the DOAP system to build input files for the REPEAT charge fitting program that reflects a need to assess the ESP grid with periodic considerations or not.

REPEAT_VDW_FACTOR [X]**:**

The REPEAT_VDW_FACTOR directive instructs the DOAP system to build input files for the REPEAT charge fitting program that includes a scaling factor for balancing the van der Waals potential with the electrostatic potential being fit to the ESP.

REPEAT_RESP_PENALTIES [0/1]:

The REPEAT_RESP_PENALTIES directive instructs the DOAP system to build input files for the REPEAT charge fitting program that directs REPEAT to use a RESP

penalty function for the charge fitting. This is useful when there are deeply buried atoms in the framework, as the general fitting method only tries to fit charges to the ESP outside the van der Waals accessible area of the molecule and sometimes comes up with very unphysical charges for buried atoms.

REPEAT_READ_CUTOFF [0/1]**:**

The REPEAT_READ_CUTOFF directive instructs the DOAP system to build an input file for the REPEAT charge fitting program that directs REPEAT to read the Ewald summation charge cutoff flag.

REPEAT_R_CUTOFF [X]**:**

The REPEAT_R_CUTOFF directive instructs the DOAP system to build an input file for the REPEAT charge fitting program specifying a cutoff distance for the Ewald summation component of the charge fitting process.

REPEAT_APPLY_SYMMETRY_RESTRAIN [0/1]**:**

The REPEAT_APPLY_SYMMETRY_RESTRAIN directive instructs the DOAP system to generate input files for the REPEAT charge fitting program which tell REPEAT to take into consideration the symmetry of the framework when assigning charges to equivalent atoms. An atom connectivity file is also supplied in this case. Currently, the DOAP system does not have the ability to assess the symmetry of the supplied framework and cannot generate an atom connectivity file for use with REPEAT.

REPEAT_USE_GODDARD [0/1]:

The REPEAT_USE_GODDARD directive instructs the DOAP system to create an input file for the REPEAT charge fitting program that specifies it should use Goddard-type restraints in the penalty function when fitting charges. The Goddard-type weights scale the electronegativity and self Coulombic interaction components of the restraining term of the penalty function.

REPEAT_GODDARD_WEIGHT [X]**:**

The REPEAT_GODDARD_WEIGHT directive instructs the DOAP system to create an input file for the REPEAT charge fitting program that specifies the weight which should be used for the Goddard-type restraints used in the penalty function for the charge fitting.

GCMC_SUPERCELL_DIM_[X] [N] **:**

The GCMC_SUPERCELL_DIM_[X] directive instructs the DOAP system to generate a super-cell for the GCMC simulation with N copies of the original unit cell in the [X] dimension. This directive is specified 3 times in the DOAP input, once for each of the dimensions of Cartesian space.

GCMC_GUEST_FILE [FILE_PATH]**:**

The GCMC_GUEST_FILE directive specifies the path location of the GUEST file that the DOAP system will use to compile the inputs and coordinate systems for the GCMC simulation.

GCMC_CUTOFF [X] :

The GCMC_CUTOFF directive instructs the DOAP system to construct GCMC simulation input files for the system with a force cut off of X angstroms.

GCMC_DELR [X]:

The GCMC_DELR directive instructs the DOAP package to construct GCMC simulation input files with a Verlet neighbour list including atoms in an X Angstrom radius.

GCMC_EWALD_PRECISION [PRECISION]**:**

The GCMC_EWALD_PRECISION directive instructs the DOAP system to construct GCMC input files that specify the level of precision for Ewald summation of electrostatic forces.

GCMC_MOL_TYPES [N]:

The GCMC_MOL_TYPES directive instructs the DOAP system to construct GCMC simulation input files for X number of molecules. Currently, there are always two

molecules, the framework and the guest, however the option to include more is left open should multiple guest handling be implemented in the future.

GCMC_IMCON [N]:

The GCMC_IMCON directive instructs the DOAP system to construct GCMC simulation input files which reflect a parallelepiped unit cell structure.

FASTMC_NUM_GUESTS [N]:

The FASTMC_NUM_GUESTS directive instructs the DOAP system to construct *fastmc* input files which specify that the system has X number of guests.

FASTMC_GUEST_PP [X]:

The FASTMC_GUEST_PP directive instructs the DOAP system to construct *fastmc* input files which specify a partial pressure for the guest gas in the framework of X atmospheres.

FASTMC_SIM_LENGTH [N]:

The FASTMC_SIM_LENGTH directive instructs the DOAP system to construct *fastmc* input files which specify that the simulation should last X number of non-equilibration steps over all branches of the simulation.

FASTMC_EQ_TIME [N]:

The FASTMC_EQ_TIME directive instructs the DOAP system to construct *fastmc* input files which specify that the simulation should be allowed to equilibrate for X number of GCMC steps.

Overall, these components of the software allow the user to harness its utility and also appreciate the nature of the development of the user side abstraction to the code, should they seek to extend its functionality.

Chapter 4 - Implementation and Computational Details

Chapter 4 provides a lower level description of the individual components of the software, providing some rationale for the use of certain elements in the code and a technical understanding of the specific details of each module and library. In addition, and example execution of the software is described to provide some expectation of the user experience.

4.1 - Programming Devices Used

In order to explain the details surrounding the implementation of the DOAP, certain devices used in the development of the software are first described.

-The NullJob Class:

The NullJob class is a general job submission class that contains all the methods common to any queued job object. Any new type of job that can be submitted to the queuing system can then be implemented simply by declaring a class which is a composition of the NullJob class. Only the “run” method need be defined in the new object to execute the specific program for the job type being created.

The NullJob class includes the following methods:

– _is_title_line
– _try_get_queue_list
– _get_queue_list
– _exists_on_queue
– _wait_for_completion
– run

By convention, those which are internal methods (run by other functions) are prefixed by an underscore. In the case of all job classes within the DOAP software, only the **run** method uses the **_wait_for_completion** method. All other methods are used by the **_wait_for_completion** method.

The **_wait_for_completion** method sleeps until a submitted job is no longer on the system queue. It uses the **_get_queue_list** method to get a listing of the jobs queued on the system. The presence of a specified field is checked for. On the Wooki cluster, the PBS queuing system is used, and provides a job id. The **_get_queue_list** method loops the **_try_get_queue_list** method until the queue_list is returned. It verifies that the return is valid by using the **_is_title_line** to make sure that the first line of the queuing system output is there. These methods were implemented because of problems found to be occurring on the Wooki cluster where the queuing system would not always return the list of job ids. When long simulations are run, the queuing system is polled for completion of a job many times and subject to having a bad queue list returned.

-Function Closures:

Where functions are needed that will only ever need be called by a single other function, the called function is declared inside of the calling function. This is termed a function closure. In the DOAP code base, function closures are used to improve the readability of the code, and better convey the intent of the function. Function closures

allow for the inner function to access the scope of the outer function and provide a means of simplifying potentially convoluted code structures.

-Second Order Functions:

In programming, functions that take other functions as parameters are termed second order functions.[46] Having functions structured in this way allows a single section of code to be immensely more useful. In the DOAP package, second order functions are used when validating lines of input. In this case, the secondary function, passed in to the first function as a parameter, is a filtering function, describing whether a line matches a particular syntax by some test, and returning its value to the original function. The function that is passed to the validating function (the first function) depends on the type of information DOAP is working on at any particular time. Second order functions allow DOAP to perform complex tasks, conveyed in simple form.

4.2 - Breakdown of Components

The code base of the DOAP system is organized into libraries. Each library contains tools used in the completion of a specific task, each of which pertains to a specific part of the overall gas uptake simulation.

The current version of the DOAP system was built specifically to operate on the Wooki cluster. The program is operated by way of two main executable files:

1 - submit_gas_uptake_job.py:

2 – get_gas_uptake.py:

submit_gas_uptake_job.py:

The submit_gas_uptake_job.py file is the main executable for the DOAP system. It takes two command line arguments when executed. The first argument for this executable is the name of the file which stores the input unit cell coordinates and vectors. The second argument is the desired name of the DOAP simulation job which will be submitted. Once the program is executed, it will submit to the PBS queuing system a job which will copy all of the required files to perform the overall simulation from the execution directory to a directory on the shared_scratch mount point, which is shared by all nodes on the Wooki cluster. Once the files have been copied to the node that the queuing system has assigned, it will execute the get_gas_uptake.py command which will carry the simulation through the rest of its operations. The parameters used with the submit_gas_uptake_job.py program are applied directly to the get_gas_uptake.py program. Once the get_gas_uptake.py command is finished, the submit_gas_uptake_job.py will copy the files back to the original submission point under a directory containing the job name. It is from this initially assigned node that the get_gas_uptake.py program will submit other jobs, return and compile information, and operate.

get_gas_uptake.py

The get_gas_uptake.py program is the main engine for the DOAP system. It handles the submission of other simulation jobs to the cluster's queuing system and performs any non-specific calculations involved in the overall simulation.

The general procedure of the get_gas_uptake.py program is:

- Process the configuration file
    - Import the configuration file, JOB_OPTIONS
    - Validate the configuration file
        - Validate the arguments
        - Validate the values
- Calculate the electrostatic potential grid
    - Create a CPMD input file
    - Create an instance of a CPMD job object
    - Run the CPMD job and wait for completion
- Get charges from CPMD / REPEAT
    - Get the required REPEAT input files
    - Write the REPEAT parameter file
    - Create an instance of a REPEAT job object
    - Run the REPEAT job and wait for completion
    - Import the REPEAT charge list
- Adjust charges and assign them to atoms
    - Adjust the charges to reflect the net charge
    - Get atom coordinates from CPMD output

- Relabel and reorder atom labels to match order of charges

- Get gas uptake from GCMC simulation:
  - Create super cell coordinates
  - Import van der Waals parameters from vdW file
  - Write the *fastmc* GCMC simulation input files
  - Create an instance of a *fastmc* job object
  - Run the *fastmc* job object and wait for completion

4.3 - Implementation Details

For each of the main tasks outlined above in section 4.2, a set of simpler commands is followed. In the following section, each of these tasks is broken down into greater detail. Where pertinent, function references (outlined in **bold)** and explanation are provided to illustrate the overall process.

- **Process the configuration file:**
  - **doap_tools.return_effective_config_file** takes the path of a DOAP_OPTIONS file as an argument and returns a config_dictionary that has been primitively validated. To do this:

- An uncommented line-list is generated from the raw DOAP_OPTIONS file.
- Options with the prefix “MAIN”, those outlining the very broad conditions of the simulation, are pulled from the file and stored in the main option prefix list. These options describe which software packages are being used for the overall simulation.
    - The main options are pulled to determine which packages are used and which other options should be specified, to be pulled and validated from the rest of the DOAP_OPTIONS file. eg. If the simulation is using SIESTA, the CPMD options need not be used (or valid) if present in the DOAP_OPTIONS file.
    - The “MAIN” prefixed options are pulled from the file. The shape of the resulting data structure is verified and the information is stored in a dictionary.
    - The dictionary is sent to **data_validation.validate_config_arguments** for verification of its values against a castable type and syntax regular expression comparison, further validating that the information is not incorrect.
- The prefixes “GENERAL”, and “GCMC” are added to the main option prefix list to account for options which are always used in the simulation.
- All options included in names in the main option prefix list are added to a list, which in turn is validated against shape, stored in a dictionary, validated against type and regular expression through the

data_validation.py library and stored in a configuration dictionary variable which is returned.

- The configuration dictionary is further validated:
  - All file and path information from the options in the configuration dictionary are verified to be valid and exist. This serves to resolve/report any file connectivity issues that the node is having should there be something awry with the configuration of the system that DOAP is being run on.
  - The cell cutoff distances specified in the configuration dictionary are checked against the unit cell vectors to make sure that they are less than 1/2 the minimum distance of the cell.
  - The structure and composition of the unit cell coordinate file is checked:
    - Each atom coordinate line must be formatted correctly.
    - Three unit cell vector lines must be present and have the appropriate format.
    - All atom labels must be resolvable. Unresolvable label names will prevent the DOAP system from constructing the requisite files for certain simulations properly. eg. “Qb” cannot be identified as any known element and will fail at this point, exiting the DOAP system.
    - A warning is raised if there are lines present that do not include atom or unit cell vector lines.

- **Calculate the electrostatic potential grid:**
  - Create a CPMD input file:
    - **cpmd_tools.create_cpmd_input_file** takes 3 parameters; the xyz input file name, the desired CPMD input file name, and the charge configuration dictionary which is a subset of the more general configuration dictionary already constructed. The **cpmd_tools.create_cpmd_input_file** function accomplishes two things. First, it generates a file [CPMD input file name].in and secondly records the order of the original atomic coordinates in a file named [CPMD input file name].atom_order. This allows the atom coordinates and charges to be returned to their original order once the charges are finally calculated. The atoms become disordered because CPMD requires that the atoms be laid out in the &ATOM section of the CPMD input file by atom type. That is, all carbon atoms are in one section of the input file, all hydrogen atoms in another, and so forth.
    - The **cpmd_tools.create_cpmd_input_file** function performs its task with the following procedure:
      - **xyz_tools.parse_unit_cell_file** is called to parse the xyz input file, providing unit cell vectors and coordinates.
      - **xyz_tools.reassign_proper_atom_labels** is called to properly assign periodic table style atom labels to atoms in the unit cell. It accomplishes this by trying to make the best match on a character-iterative basis for each atom label in the xyz unit cell file.

- An atom label list is created to determine the minimal set of atom labels and the quantity of each atom type for iteration purposes.
    - Hydrogen is reordered to end of the atom label list. In the event that the hydrogen atoms are to be geometrically optimized, CPMD requires that they are at the end of the file.
- A set of pseudo-potential files are selected for the atoms found in the framework.
    - The list of available pseudo-potential files are pulled from the path originally specified in the DOAP_OPTIONS files, now in the verified charge configuration dictionary.
    - **cpmd_tools.confirm_psp_choices** takes the default functional and pseudo-potential type from the charge configuration dictionary and variations of the expected associated file name is checked for in the pseudo-potential file list.
    - Should there be no match, **cpmd_tools.select_new_psp_file** is called to select an appropriate file.
        - A list of appropriate pseudo-potential files for the selected atom type are displayed and index. The user is asked to choose a value corresponding the desired pseudo-potential file.
- Each pseudo-potential file is characterized as a BINARY or FORMATTED PSP file.
    - **cpmd_tools.assign_binary_or_formatted_to_psps** is sent the list of accepted PSP files.

        - **cpmd_tools.assign_binary_or_formatted_psps** calls **is_binary** to assess if each file is binary or not by taking a sample of the first 2048 characters of the file and determining if more than 80 percent of them are characters outside of the alphanumeric range.
- **xyz_tools.index_and_sort_coordinate_list** is sent the atom coordinates and list of atom types. It returns a list of ordered atoms, sorted by atom type, and the index list which serves as a record of the original order of the atoms before being sorted by atom type.
- **write_cpmd_input_file** is called, which in turn calls functions to write each distinct section of the CPMD input file. They are:

    - The **&CPMD** section:
        - The **write_cpmd_section** function constructs a &CPMD section string including the options associated with this section of the input.
            - Convergence criteria for the orbitals.
            - Maximum iterations for the convergence.
        - In the event that hydrogen optimization has been selected and the system contains hydrogen atoms, the geometry optimization directive is also added.
        - The &CPMD section string is written to file.

- The **&DFT** section:
  - The **write_dft_section** function constructs a &DFT section string including options associated with this section of the input.
    - The functional used in the calculation.
  - The &DFT section string is written to file.

- The **&SYSTEM** section:
  - The **write_system_section** function constructs a &SYSTEM section string including options associated with this section of the input.
    - The plane wave energy cutoff.
    - The charge of the system.
    - The ratio between the wave function energy cut off and the density cut off. The “DUAL” parameter.
  - The PSP choices are checked for Vanderbilt ultra-soft pseudo-potentials.
    - Should they exist, the DUAL directive is included in the &SYSTEM section to allow for better geometry optimization should the option be selected.
  - The &SYSTEM section string is written to file.

- The **&ATOMS** section

        - The **write_atom_section** function constructs an &ATOMS section string including options associated with this section of the CPMD input.
            - Maximum l-quantum number.
            - Atom quantities and coordinates.
            - Associated PSP files.
        - Subsections for each atom in the atom list are created, specifying the pseudo-potential file, the maximum l-quantum number, and number of atoms of that type. This is followed by the atom coordinates for each atom of that type.
        - Each atom type is iterated through.
        - The hydrogen atom section is left to the end should there be any.
        - The non-hydrogen atoms are set to be fixed, should a geometry optimization be selected for in the options, leaving only the hydrogen atoms to be optimized.
    - The order of the atom label sorted coordinates is recorded to file.

- Create an instance of a CPMD job object, run the CPMD job and wait for completion
    - A **queue_tools.CpmdJob** object is created. The CpmdJob class is a composition of the NullJob class and has only one method of its own,

run. The **run** method executes the job on the queue and invokes internal methods from the NullJob class to wait for it to finish.

- **Get charges from CPMD / REPEAT:**
  - Get the required REPEAT input files.
    - The **repeat_tools.get_repeat_input_files** function copies the resulting files from the CPMD job temporary directory to the DOAP job directory on the main DOAP node.
    - The function converts the electrostatic potential grid file named “ELPOT” from reciprocal space coordinates to real space coordinates using the cpmd2cube.x program that comes with CPMD.
  - Write the REPEAT parameter file.
    - The **repeat_tools.write_repeat_parameter_file** function creates the “REPEAT_param.inp” file which is needed to perform a REPEAT charge fitting.
    - The REPEAT_param.inp” file is composed of “REPEAT_” prefixed options from the DOAP_OPTIONS file.
  - Instatiate a REPEAT job object, run the REPEAT job and wait for completion
    - A **queue_tools.RepeatJob** object is created. The RepeatJob class is a composition of the NullJob class and has only one method of its own, **run.** The **run** method executes the job on the queue and invokes internal methods from the NullJob class to wait for it to finish.

- Import the REPEAT charge list.
  - The **repeat_tools.import_repeat_charge_file** function is called to parse the lines in the REPEAT charge output, and provide the charge for each atom in the fitting.

- **Adjust charges and assign them to atoms:**
  - Adjust the charges to reflect the net charge.
    - The **molecular_mechanics.adjust_charges_to_match_net_charge** function takes a list of charges, the net charge and a degree of specification (the value to adjust by in correcting the charge) and iterates through the charge list adjusting each atom by the adjustment value until the net charge is reflected in the sum of the partial charges.
  - Get atom coordinates from CPMD output.
    - The **cpmd_tools.parse_cpmd_geometry_file** function takes the location of a CPMD generated “GEOMETRY.xyz” file and parses out the atom names and coordinates into a list.
  - Relabel and reorder atom labels to match order of charges.
    - The **xyz_tools.import_order_list** function import the order list previously generated from file.
    - The **molecular_mechanics.reorder_charges** function takes the charge list and order list as parameters and reorders the charges so that they reflect the order found in the original input file.

        - The **xyz_tools.reorder_coordinates** function takes the coordinate list and order list as parameters and reorders the coordinates so that they reflect the order found in the original input file.

- Get gas uptake from GCMC simulation:
    - Create supercell coordinates.
        - The **xyz_tools.create_supercell** function takes a set of unit cell coordinates, the unit cell vectors and an array of supercell dimensions as inputs and returns vectors and coordinates pertaining the resulting supercell. The following process is used to create a supercell:
            - A list of “vector masks” for all combinations of the supercell dimensions is created.
                - The vector mask depicts the multiples of the unit cell vectors which need to be added to the unit cell coordinates.
                - eg. for a 2x2x2 super cell, the vector masks would be:

                  [0,0,0],[0,0,1],[0,1,0],[0,1,1],[1,0,0],[1,0,1],[1,1,0],[1,1,1]
            - Each of the vector masks are multiplied by the unit cell vectors to create a set of translation vectors for the unit cell coordinates.
            - The translations vectors are applied to the unit cell coordinates to create all the image cells of the super cell.
            - The unit cell vectors are multiplied by the super cell dimensions to make the super cell unit vectors.

- Import van der Waals parameters from vdW file.
  - The **molecular_mechanics.import_vdw_values** function pulls in the van der Waals Lennard-Jones parameters.
- Write the *fastmc* GCMC simulation input files.
  - The **fastmc_tools.write_fastmc_simulation_files** function generates all the necessary input files to run a *fastmc* GCMC simulation. It includes three functions. They are:
    - **write_fastmc_config_file:** Constructs a *fastmc* CONFIG file, containing the atomic coordinates.
    - **write_fastmc_control_file:** Constructs a *fastmc* CONTROL file, containing the directives for the simulation, i.e. temperature of simulation, number of equilibration/ non-equilibration steps, *etc.*
    - **write_fastmc_field_file:** Constructs a *fastmc* FIELD file, containing all of the force field parameters and definition of molecules within the simulation.
- Create an instance of a *fastmc* job object, run the *fastmc* job and wait for completion.
  - A **queue_tools.FastmcJob** object is created. The FastmcJob class is a composition of the NullJob class and has only one method of its own, **run.** The **run** method executes the job on the queue and invokes internal methods from the NullJob class to wait for it to finish.

4.4 - Termination of the GCMC simulation

The intent of the Grand Canonical Monte Carlo simulation is to determine the equilibrium amount of gas that can be adsorbed in a framework. Molecules of gas are inserted, deleted and perturbed in the super cell in order to find preferential low-energy states of the overall system. Unlike molecular dynamics, Monte Carlo methods sample randomly and are said to be based on chance. That is to say that they have no causal direction. The termination of the simulation must be based on the convergence of the property being observed. In the case of the Grand Canonical Monte Carlo simulation, the number of guest particles is this observable. As we will see in the next Chapter, a range of metrics exists for characterizing the "converged" end state of a GCMC simulation.

4.5 - Example Execution of the Program

In an example execution, ZIF-8 crystallographic data were downloaded from the Cambridge Crystallographic Data Centre[55], cleaned of any non-framework atoms and converted into an xyz format file. The file was placed in a directory containing a copy of the DOAP software package. The DOAP_OPTIONS file was edited to perform a simulation 6000000 GCMC steps long, equilibrating over 300000 GCMC steps. Hydrogen atoms were not set to be optimized. The package was executed using the command "./get gasuptake INPUT_ZIF8 zif_8_testrun".

Moments later the software outputs a simple description of the system;

System described as: ['Zn', 'C', 'N', 'H'] {'H': 120, 'Zn': 12, 'C': 96, 'N': 48}

and mentions the location of the pseudo-potential file path that is selected and the psp files that have been chosen given the psp type and the functional type specified in the DOAP_OPTIONS file. The appropriate CPMD input file is generated for the system stored in the XYZ file and the CPMD job is sent to the queue. One hour and fifty three minutes later, the program finished, the output files are verified for completion and the software generates a REPEAT input file and submits the charge fitting job to the queue. The REPEAT job finished approximately 25 minutes later, providing its output in the REPEAT_ESP.esp_fit.out file. The file is parsed for charge values, assigned to appropriate atoms, and the program proceeds to generate the FIELD, CONFIG, and CONTROL file before submitting a *fastmc* job to the queue to evaluate the gas uptake of the ZIF-8 framework. The *fastmc* program creates a single branch01 directory containing a numguests.out file. The *fastmc* software takes approximately 18 hours to perform 6 million *fastmc* GCMC steps, after which, the numguests.out file is done being written to. Below is a graphical depiction of the uptake data returned from the simulation of ZIF8.

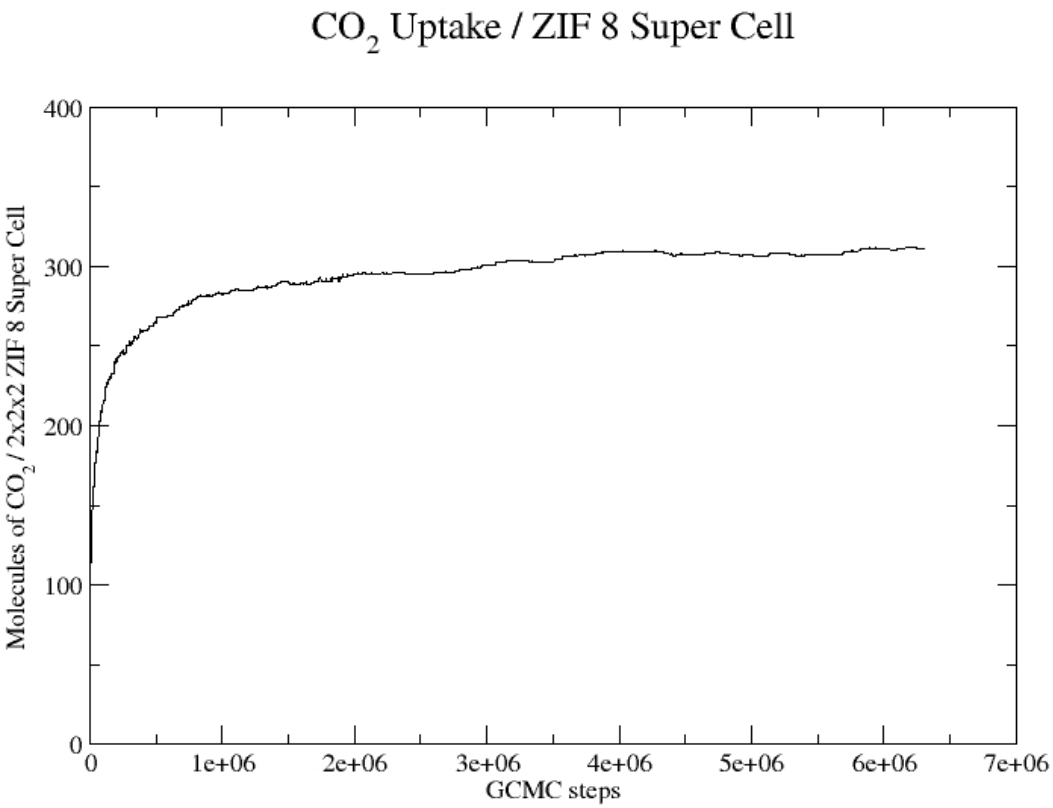


Figure 4.1 Uptake of CO_2 gas molecules per 2x2x2 super cell of a ZIF-8 framework.

Overall, the simulation package is able to generate to required inputs and execute all the relevant simulations in a straightforward manner, allowing for the automation of the acquisition of gaseous uptake information, a result which otherwise would be potentially fraught with human error and take a considerably longer amount of time to produce. In addition chapter 4 has outlined the complex nature of the code, with its many components juxtaposed the simplicity and straightforward nature of its implementation.

4.6 – Comparison of Automated and Non-Automated Simulation Work Flow

To ensure the DOAP software does not bias or cause deleterious effects to the simulation process a comparison was done between the automated simulation process and a manual “by hand” simulation process. Two frameworks of broadly differing composition were evaluated, the ZIF-8 framework, and sodalite, a naturally occurring zeolite structure. Below, the “by hand” and automated simulation processes are compared with respect to the charges produced (table 4.1, table 4.2), the field parameters (tables 4.3 – 4.6) and further, the gas uptake graphs (figure 4.2, figure 4.3) are contrasted against each other showing qualitatively consistent uptake information.

Table 4.1 Charges of atoms in a sodalite framework generated by the REPEAT method, instantiated "by hand" (HAND) and automatically, using the DOAP software (DOAP). The charges are specifically identical showing that the DOAP software performs as expected in this capacity.

Atom #	Atom Type	Sodalite	
		HAND	DOAP
1	Si	1.34073160767995	1.34073160767995
2	Si	1.34073193857913	1.34073193857913
3	Si	1.34072688521891	1.34072688521891
4	Si	1.34072718272534	1.34072718272534
5	Si	1.34073204717144	1.34073204717144
6	Si	1.34072686024014	1.34072686024014
7	Si	1.34072719372787	1.34072719372787
8	Si	1.34073163502366	1.34073163502366
9	Si	1.34073200472144	1.34073200472144
10	Si	1.34072727776504	1.34072727776504
11	Si	1.34072730440614	1.34072730440614
12	Si	1.34073208045351	1.34073208045351
13	O	-0.670366730895893	-0.670366730895893
14	O	-0.670366875190793	-0.670366875190793
15	O	-0.670364366215308	-0.670364366215308
16	O	-0.670364501632504	-0.670364501632504
17	O	-0.670367901515331	-0.670367901515331
18	O	-0.670368004359882	-0.670368004359882
19	O	-0.670366748418845	-0.670366748418845
20	O	-0.670366868419581	-0.670366868419581
21	O	-0.670366737252123	-0.670366737252123
22	O	-0.670364312329807	-0.670364312329807
23	O	-0.670366699745038	-0.670366699745038
24	O	-0.670367896855578	-0.670367896855578
25	O	-0.670363177967410	-0.670363177967410
26	O	-0.670363331636151	-0.670363331636151
27	O	-0.670363558295975	-0.670363558295975
28	O	-0.670363731506231	-0.670363731506231
29	O	-0.670362026247503	-0.670362026247503
30	O	-0.670362210323782	-0.670362210323782
31	O	-0.670363176262180	-0.670363176262180
32	O	-0.670363347558150	-0.670363347558150
33	O	-0.670363137520289	-0.670363137520289
34	O	-0.670363517037429	-0.670363517037429
35	O	-0.670363160895702	-0.670363160895702
36	O	-0.670361999631082	-0.670361999631082

Table 4.2 Charges of atoms in a ZIF 8 framework generated by the REPEAT method, instantiated “by hand” (HAND) and automatically, using the DOAP software (DOAP). The charges are specifically identical showing that the DOAP software performs as expected in this capacity.

Atom #	Atom Type	ZIF 8	
		HAND	DOAP
1	Zn	0.570852665223576	0.570852665223576
2	Zn	0.589745045719203	0.589745045719203
3	Zn	0.596968346503828	0.596968346503828
4	Zn	0.587149204424635	0.587149204424635
5	Zn	0.583849743410194	0.583849743410194
6	Zn	0.615430505897440	0.615430505897440
7	Zn	0.587810583565587	0.587810583565587
8	Zn	0.586744598593247	0.586744598593247
9	Zn	0.567669524255163	0.567669524255163
10	Zn	0.600605637887940	0.600605637887940
11	Zn	0.567770899491336	0.567770899491336
12	Zn	0.570509777600156	0.570509777600156
13	C	-0.593829777754851	-0.593829777754851
14	C	0.405035580141666	0.405035580141666
15	C	-0.571270520369522	-0.571270520369522
16	C	-0.212527880472175	-0.212527880472175
17	C	-0.206243044003929	-0.206243044003929
18	C	-0.242311534828232	-0.242311534828232
19	C	0.400623191888071	0.400623191888071
20	C	0.400708625134570	0.400708625134570
21	C	-0.234457747834254	-0.234457747834254
22	C	0.393649330992574	0.393649330992574
23	C	0.406566873930021	0.406566873930021
24	C	-0.214815711844170	-0.214815711844170
25	C	-0.234194109654132	-0.234194109654132
26	C	-0.562408883249987	-0.562408883249987
27	C	-0.577628151574210	-0.577628151574210
28	C	0.404340653433165	0.404340653433165
29	C	-0.231854794506375	-0.231854794506375
30	C	-0.232677287520447	-0.232677287520447
31	C	-0.553491481905585	-0.553491481905585
32	C	0.391509068470019	0.391509068470019
33	C	-0.207083710608790	-0.207083710608790
34	C	-0.586803109413067	-0.586803109413067
35	C	-0.231802215877619	-0.231802215877619
36	C	-0.568001415893529	-0.568001415893529

37	C	-0.576781754215440	-0.576781754215440
38	C	0.397354783381133	0.397354783381133
39	C	0.396924585538231	0.396924585538231
40	C	0.390860083172567	0.390860083172567
41	C	-0.246333632419633	-0.246333632419633
42	C	-0.233169973410334	-0.233169973410334
43	C	0.400725695737018	0.400725695737018
44	C	-0.219970867857260	-0.219970867857260
45	C	-0.237513167993326	-0.237513167993326
46	C	-0.246527487572902	-0.246527487572902
47	C	0.397566653364782	0.397566653364782
48	C	-0.583957558526109	-0.583957558526109
49	C	-0.217763918904483	-0.217763918904483
50	C	-0.253902200487618	-0.253902200487618
51	C	-0.213342530087176	-0.213342530087176
52	C	0.396737941259206	0.396737941259206
53	C	-0.582060836547048	-0.582060836547048
54	C	-0.585539788958708	-0.585539788958708
55	C	-0.574178301389431	-0.574178301389431
56	C	-0.221969708654739	-0.221969708654739
57	C	0.395499467246061	0.395499467246061
58	C	-0.583969561041734	-0.583969561041734
59	C	-0.224148663956075	-0.224148663956075
60	C	-0.201299082445906	-0.201299082445906
61	C	-0.223461706035039	-0.223461706035039
62	C	-0.191850080157245	-0.191850080157245
63	C	-0.234583846157528	-0.234583846157528
64	C	0.393324728342785	0.393324728342785
65	C	0.403304155484993	0.403304155484993
66	C	-0.195895660003335	-0.195895660003335
67	C	-0.254808416277191	-0.254808416277191
68	C	-0.223117554861529	-0.223117554861529
69	C	-0.222294181777678	-0.222294181777678
70	C	-0.216074227798155	-0.216074227798155
71	C	-0.214055243339103	-0.214055243339103
72	C	-0.238249357903904	-0.238249357903904
73	C	-0.578051840777740	-0.578051840777740
74	C	-0.239296527955264	-0.239296527955264
75	C	-0.246463376202195	-0.246463376202195
76	C	0.401638682602720	0.401638682602720
77	C	0.394300232876897	0.394300232876897
78	C	-0.558870293940517	-0.558870293940517
79	C	-0.213009175367542	-0.213009175367542
80	C	0.402475633055392	0.402475633055392
81	C	-0.587365431836376	-0.587365431836376
82	C	-0.587367096359547	-0.587367096359547
83	C	-0.213655107784705	-0.213655107784705
84	C	-0.195945618823597	-0.195945618823597
85	C	-0.577471495674897	-0.577471495674897

86	C	-0.581410281876950	-0.581410281876950
87	C	-0.227512663760617	-0.227512663760617
88	C	-0.569137690788423	-0.569137690788423
89	C	-0.230420992028675	-0.230420992028675
90	C	-0.236654627489347	-0.236654627489347
91	C	-0.563938339022446	-0.563938339022446
92	C	-0.200681841252775	-0.200681841252775
93	C	-0.556408081296102	-0.556408081296102
94	C	-0.223623366893128	-0.223623366893128
95	C	-0.203274186376823	-0.203274186376823
96	C	-0.248407436814776	-0.248407436814776
97	C	0.388667893861340	0.388667893861340
98	C	-0.566426723790734	-0.566426723790734
99	C	-0.559903947708762	-0.559903947708762
100	C	-0.205367238587569	-0.205367238587569
101	C	-0.216232579413820	-0.216232579413820
102	C	0.400897564177370	0.400897564177370
103	C	-0.219221056476354	-0.219221056476354
104	C	0.405795163028085	0.405795163028085
105	C	-0.243685019576536	-0.243685019576536
106	C	0.405047219548141	0.405047219548141
107	C	0.388894684324992	0.388894684324992
108	C	-0.253758240133901	-0.253758240133901
109	N	-0.245957154430717	-0.245957154430717
110	N	-0.247386342495528	-0.247386342495528
111	N	-0.253148813954403	-0.253148813954403
112	N	-0.241165532762492	-0.241165532762492
113	N	-0.260685456733257	-0.260685456733257
114	N	-0.257359195076736	-0.257359195076736
115	N	-0.244449392916618	-0.244449392916618
116	N	-0.252556076977270	-0.252556076977270
117	N	-0.266534007928608	-0.266534007928608
118	N	-0.256948151820586	-0.256948151820586
119	N	-0.238070305001392	-0.238070305001392
120	N	-0.255685649802445	-0.255685649802445
121	N	-0.249205292056851	-0.249205292056851
122	N	-0.251307204625260	-0.251307204625260
123	N	-0.249615999528291	-0.249615999528291
124	N	-0.244449599981266	-0.244449599981266
125	N	-0.251979360628368	-0.251979360628368
126	N	-0.240311027248270	-0.240311027248270
127	N	-0.240685821706723	-0.240685821706723
128	N	-0.251436728551275	-0.251436728551275
129	N	-0.265341397324913	-0.265341397324913
130	N	-0.240918226584740	-0.240918226584740
131	N	-0.242979335874927	-0.242979335874927
132	N	-0.251741716176604	-0.251741716176604
133	N	-0.251475559232536	-0.251475559232536
134	N	-0.257544764532753	-0.257544764532753
135	N	-0.271987566845905	-0.271987566845905

136	N	-0.256531903863773	-0.256531903863773
137	N	-0.246142761310325	-0.246142761310325
138	N	-0.251321110710665	-0.251321110710665
139	N	-0.237836436952336	-0.237836436952336
140	N	-0.257876252984151	-0.257876252984151
141	N	-0.241533477143618	-0.241533477143618
142	N	-0.260765830375991	-0.260765830375991
143	N	-0.248402297291556	-0.248402297291556
144	N	-0.267893789776815	-0.267893789776815
145	N	-0.268967318944366	-0.268967318944366
146	N	-0.249392852717094	-0.249392852717094
147	N	-0.254402890398793	-0.254402890398793
148	N	-0.252897750781913	-0.252897750781913
149	N	-0.264973510941527	-0.264973510941527
150	N	-0.258161625329039	-0.258161625329039
151	N	-0.249784083249953	-0.249784083249953
152	N	-0.245670728450592	-0.245670728450592
153	N	-0.257733229055200	-0.257733229055200
154	N	-0.260703537755630	-0.260703537755630
155	N	-0.241514874481141	-0.241514874481141
156	N	-0.263334083330762	-0.263334083330762
157	H	0.172402781987557	0.172402781987557
158	H	0.182528331722689	0.182528331722689
159	H	0.165958931203671	0.165958931203671
160	H	0.177084523148258	0.177084523148258
161	H	0.182500913900560	0.182500913900560
162	H	0.175819089172446	0.175819089172446
163	H	0.143866495975120	0.143866495975120
164	H	0.172323895287573	0.172323895287573
165	H	0.175191816211281	0.175191816211281
166	H	0.178362948505812	0.178362948505812
167	H	0.140656989848218	0.140656989848218
168	H	0.184653099248931	0.184653099248931
169	H	0.165481168195594	0.165481168195594
170	H	0.170236586638364	0.170236586638364
171	H	0.176716402815127	0.176716402815127
172	H	0.161212428079955	0.161212428079955
173	H	0.187726656399229	0.187726656399229
174	H	0.136859405699675	0.136859405699675
175	H	0.150376396295453	0.150376396295453
176	H	0.171543010498496	0.171543010498496
177	H	0.143105047942107	0.143105047942107
178	H	0.177760085250920	0.177760085250920
179	H	0.188923010400385	0.188923010400385
180	H	0.162376705829298	0.162376705829298
181	H	0.177239551299022	0.177239551299022
182	H	0.175448325696217	0.175448325696217
183	H	0.156859956080123	0.156859956080123
184	H	0.185829376162302	0.185829376162302
185	H	0.173172704113391	0.173172704113391

186	H	0.174874774618725	0.174874774618725
187	H	0.188794230866606	0.188794230866606
188	H	0.140018735601959	0.140018735601959
189	H	0.135232406392153	0.135232406392153
190	H	0.178088478296232	0.178088478296232
191	H	0.139694131563461	0.139694131563461
192	H	0.139821638978113	0.139821638978113
193	H	0.170919076729150	0.170919076729150
194	H	0.162685051391431	0.162685051391431
195	H	0.177606809264506	0.177606809264506
196	H	0.182079936961325	0.182079936961325
197	H	0.188454189345943	0.188454189345943
198	H	0.180072726576788	0.180072726576788
199	H	0.183199832662743	0.183199832662743
200	H	0.140211122979777	0.140211122979777
201	H	0.181055171572247	0.181055171572247
202	H	0.174985806976564	0.174985806976564
203	H	0.177706249750508	0.177706249750508
204	H	0.179332859286705	0.179332859286705
205	H	0.181235378988007	0.181235378988007
206	H	0.178591168727324	0.178591168727324
207	H	0.135422031443257	0.135422031443257
208	H	0.188523865328060	0.188523865328060
209	H	0.179853832347492	0.179853832347492
210	H	0.180803911950525	0.180803911950525
211	H	0.139791270520987	0.139791270520987
212	H	0.185929292877292	0.185929292877292
213	H	0.180612388813743	0.180612388813743
214	H	0.143187189104859	0.143187189104859
215	H	0.190010645220947	0.190010645220947
216	H	0.178616787234026	0.178616787234026
217	H	0.173202728524625	0.173202728524625
218	H	0.179276287762604	0.179276287762604
219	H	0.189817667079873	0.189817667079873
220	H	0.182752427585653	0.182752427585653
221	H	0.164767048203478	0.164767048203478
222	H	0.179011626594105	0.179011626594105
223	H	0.132515355290976	0.132515355290976
224	H	0.173466463626787	0.173466463626787
225	H	0.161953263662412	0.161953263662412
226	H	0.174258587550103	0.174258587550103
227	H	0.157829880931975	0.157829880931975
228	H	0.174397271978143	0.174397271978143
229	H	0.191120487600820	0.191120487600820
230	H	0.177336266198130	0.177336266198130
231	H	0.140314302572956	0.140314302572956
232	H	0.176319392933373	0.176319392933373
233	H	0.179459765257004	0.179459765257004
234	H	0.138876821105119	0.138876821105119

235	H	0.178628422687718	0.178628422687718
236	H	0.185521224938991	0.185521224938991
237	H	0.162971128952565	0.162971128952565
238	H	0.188417742389839	0.188417742389839
239	H	0.184719659679092	0.184719659679092
240	H	0.163485210242160	0.163485210242160
241	H	0.154549209998702	0.154549209998702
242	H	0.185994634551639	0.185994634551639
243	H	0.153914563369797	0.153914563369797
244	H	0.154685596795383	0.154685596795383
245	H	0.137607576500609	0.137607576500609
246	H	0.133200041358184	0.133200041358184
247	H	0.140161211106012	0.140161211106012
248	H	0.175155519206403	0.175155519206403
249	H	0.174704508182500	0.174704508182500
250	H	0.190779350937208	0.190779350937208
251	H	0.160557027021613	0.160557027021613
252	H	0.153267405639070	0.153267405639070
253	H	0.157063176657118	0.157063176657118
254	H	0.181769836530647	0.181769836530647
255	H	0.153132511875957	0.153132511875957
256	H	0.183163310011753	0.183163310011753
257	H	0.191294532075791	0.191294532075791
258	H	0.185190834773772	0.185190834773772
259	H	0.163033475129709	0.163033475129709
260	H	0.140833355534711	0.140833355534711
261	H	0.141064600412347	0.141064600412347
262	H	0.179246850741298	0.179246850741298
263	H	0.168067449811378	0.168067449811378
264	H	0.134442682048534	0.134442682048534
265	H	0.171638917268775	0.171638917268775
266	H	0.135450003729826	0.135450003729826
267	H	0.156906575475418	0.156906575475418
268	H	0.153552656674188	0.153552656674188
269	H	0.174783695031309	0.174783695031309
270	H	0.133693430408477	0.133693430408477
271	H	0.186128065451127	0.186128065451127
272	H	0.161233545034765	0.161233545034765
273	H	0.174286835655687	0.174286835655687
274	H	0.178524556334340	0.178524556334340
275	H	0.134286949838944	0.134286949838944
276	H	0.177580816678767	0.177580816678767

Table 4.3 Field parameters of atoms in a sodalite framework generated "by hand" (HAND) and automatically using the DOAP software (DOAP). The values are specifically identical showing that the DOAP software performs as expected in this capacity.

Atom #	Atom Type	Mass		Charge	
		HAND	DOAP	HAND	DOAP
1	O	15.9994	15.9994	-0.6704	-0.6704
2	O	15.9994	15.9994	-0.6704	-0.6704
3	O	15.9994	15.9994	-0.6704	-0.6704
4	O	15.9994	15.9994	-0.6704	-0.6704
5	O	15.9994	15.9994	-0.6704	-0.6704
6	O	15.9994	15.9994	-0.6704	-0.6704
7	O	15.9994	15.9994	-0.6704	-0.6704
8	O	15.9994	15.9994	-0.6704	-0.6704
9	O	15.9994	15.9994	-0.6704	-0.6704
10	O	15.9994	15.9994	-0.6704	-0.6704
11	O	15.9994	15.9994	-0.6704	-0.6704
12	O	15.9994	15.9994	-0.6704	-0.6704
13	O	15.9994	15.9994	-0.6704	-0.6704
14	O	15.9994	15.9994	-0.6704	-0.6704
15	O	15.9994	15.9994	-0.6704	-0.6704
16	O	15.9994	15.9994	-0.6704	-0.6704
17	O	15.9994	15.9994	-0.6704	-0.6704
18	O	15.9994	15.9994	-0.6704	-0.6704
19	O	15.9994	15.9994	-0.6704	-0.6704
20	O	15.9994	15.9994	-0.6704	-0.6704
21	O	15.9994	15.9994	-0.6704	-0.6704
22	O	15.9994	15.9994	-0.6704	-0.6704
23	O	15.9994	15.9994	-0.6704	-0.6704
24	O	15.9994	15.9994	-0.6704	-0.6704
25	Si	28.0855	28.0855	1.3408	1.3408
26	Si	28.0855	28.0855	1.3408	1.3408
27	Si	28.0855	28.0855	1.3408	1.3408
28	Si	28.0855	28.0855	1.3408	1.3408
29	Si	28.0855	28.0855	1.3408	1.3408
30	Si	28.0855	28.0855	1.3408	1.3408
31	Si	28.0855	28.0855	1.3408	1.3408
32	Si	28.0855	28.0855	1.3408	1.3408
33	Si	28.0855	28.0855	1.3408	1.3408
34	Si	28.0855	28.0855	1.3408	1.3408
35	Si	28.0855	28.0855	1.3408	1.3408
36	Si	28.0855	28.0855	1.3408	1.3408

Table 4.4 Field parameters of atoms in a ZIF 8 framework generated "by hand" (HAND) and automatically using the DOAP software (DOAP). The values are specifically identical showing that the DOAP software performs as expected in this capacity.

Atom #	Atom Type	Mass		Charge	
		HAND	DOAP	HAND	DOAP
1	Zn	65.3800	65.3800	0.5710	0.5710
2	C	12.0107	12.0107	-0.5938	-0.5938
3	Zn	65.3800	65.3800	0.5897	0.5897
4	C	12.0107	12.0107	0.4050	0.4050
5	H	1.0079	1.0079	0.1724	0.1724
6	C	12.0107	12.0107	-0.5713	-0.5713
7	N	14.0067	14.0067	-0.2460	-0.2460
8	H	1.0079	1.0079	0.1825	0.1825
9	H	1.0079	1.0079	0.1660	0.1660
10	H	1.0079	1.0079	0.1771	0.1771
11	H	1.0079	1.0079	0.1825	0.1825
12	H	1.0079	1.0079	0.1758	0.1758
13	C	12.0107	12.0107	-0.2125	-0.2125
14	C	12.0107	12.0107	-0.2062	-0.2062
15	N	14.0067	14.0067	-0.2474	-0.2474
16	C	12.0107	12.0107	-0.2423	-0.2423
17	C	12.0107	12.0107	0.4006	0.4006
18	H	1.0079	1.0079	0.1439	0.1439
19	H	1.0079	1.0079	0.1723	0.1723
20	H	1.0079	1.0079	0.1752	0.1752
21	H	1.0079	1.0079	0.1784	0.1784
22	H	1.0079	1.0079	0.1407	0.1407
23	C	12.0107	12.0107	0.4007	0.4007
24	C	12.0107	12.0107	-0.2345	-0.2345
25	N	14.0067	14.0067	-0.2531	-0.2531
26	N	14.0067	14.0067	-0.2412	-0.2412
27	N	14.0067	14.0067	-0.2607	-0.2607
28	N	14.0067	14.0067	-0.2574	-0.2574
29	H	1.0079	1.0079	0.1847	0.1847
30	C	12.0107	12.0107	0.3936	0.3936
31	H	1.0079	1.0079	0.1655	0.1655
32	C	12.0107	12.0107	0.4066	0.4066
33	H	1.0079	1.0079	0.1702	0.1702
34	N	14.0067	14.0067	-0.2444	-0.2444
35	H	1.0079	1.0079	0.1767	0.1767
36	H	1.0079	1.0079	0.1612	0.1612

37	C	12.0107	12.0107	-0.2148	-0.2148
38	N	14.0067	14.0067	-0.2526	-0.2526
39	C	12.0107	12.0107	-0.2342	-0.2342
40	C	12.0107	12.0107	-0.5624	-0.5624
41	H	1.0079	1.0079	0.1877	0.1877
42	C	12.0107	12.0107	-0.5776	-0.5776
43	Zn	65.3800	65.3800	0.5970	0.5970
44	H	1.0079	1.0079	0.1369	0.1369
45	H	1.0079	1.0079	0.1504	0.1504
46	H	1.0079	1.0079	0.1715	0.1715
47	C	12.0107	12.0107	0.4043	0.4043
48	H	1.0079	1.0079	0.1431	0.1431
49	C	12.0107	12.0107	-0.2319	-0.2319
50	N	14.0067	14.0067	-0.2665	-0.2665
51	C	12.0107	12.0107	-0.2327	-0.2327
52	H	1.0079	1.0079	0.1778	0.1778
53	C	12.0107	12.0107	-0.5535	-0.5535
54	N	14.0067	14.0067	-0.2569	-0.2569
55	C	12.0107	12.0107	0.3915	0.3915
56	H	1.0079	1.0079	0.1889	0.1889
57	C	12.0107	12.0107	-0.2071	-0.2071
58	H	1.0079	1.0079	0.1624	0.1624
59	H	1.0079	1.0079	0.1772	0.1772
60	H	1.0079	1.0079	0.1754	0.1754
61	C	12.0107	12.0107	-0.5868	-0.5868
62	H	1.0079	1.0079	0.1569	0.1569
63	H	1.0079	1.0079	0.1858	0.1858
64	H	1.0079	1.0079	0.1732	0.1732
65	H	1.0079	1.0079	0.1749	0.1749
66	H	1.0079	1.0079	0.1888	0.1888
67	C	12.0107	12.0107	-0.2318	-0.2318
68	H	1.0079	1.0079	0.1400	0.1400
69	N	14.0067	14.0067	-0.2381	-0.2381
70	Zn	65.3800	65.3800	0.5871	0.5871
71	C	12.0107	12.0107	-0.5680	-0.5680
72	H	1.0079	1.0079	0.1352	0.1352
73	N	14.0067	14.0067	-0.2557	-0.2557
74	N	14.0067	14.0067	-0.2492	-0.2492
75	N	14.0067	14.0067	-0.2513	-0.2513
76	H	1.0079	1.0079	0.1781	0.1781
77	C	12.0107	12.0107	-0.5768	-0.5768
78	H	1.0079	1.0079	0.1397	0.1397
79	H	1.0079	1.0079	0.1398	0.1398
80	C	12.0107	12.0107	0.3974	0.3974
81	C	12.0107	12.0107	0.3969	0.3969
82	C	12.0107	12.0107	0.3909	0.3909
83	H	1.0079	1.0079	0.1709	0.1709
84	H	1.0079	1.0079	0.1627	0.1627
85	C	12.0107	12.0107	-0.2463	-0.2463

86	N	14.0067	14.0067	-0.2496	-0.2496
87	Zn	65.3800	65.3800	0.5838	0.5838
88	C	12.0107	12.0107	-0.2332	-0.2332
89	C	12.0107	12.0107	0.4007	0.4007
90	H	1.0079	1.0079	0.1776	0.1776
91	H	1.0079	1.0079	0.1821	0.1821
92	C	12.0107	12.0107	-0.2200	-0.2200
93	H	1.0079	1.0079	0.1885	0.1885
94	C	12.0107	12.0107	-0.2375	-0.2375
95	H	1.0079	1.0079	0.1801	0.1801
96	H	1.0079	1.0079	0.1832	0.1832
97	H	1.0079	1.0079	0.1402	0.1402
98	H	1.0079	1.0079	0.1811	0.1811
99	C	12.0107	12.0107	-0.2465	-0.2465
100	Zn	65.3800	65.3800	0.6154	0.6154
101	H	1.0079	1.0079	0.1750	0.1750
102	N	14.0067	14.0067	-0.2444	-0.2444
103	C	12.0107	12.0107	0.3976	0.3976
104	C	12.0107	12.0107	-0.5840	-0.5840
105	C	12.0107	12.0107	-0.2178	-0.2178
106	N	14.0067	14.0067	-0.2520	-0.2520
107	H	1.0079	1.0079	0.1777	0.1777
108	H	1.0079	1.0079	0.1793	0.1793
109	H	1.0079	1.0079	0.1812	0.1812
110	H	1.0079	1.0079	0.1786	0.1786
111	H	1.0079	1.0079	0.1354	0.1354
112	N	14.0067	14.0067	-0.2403	-0.2403
113	H	1.0079	1.0079	0.1885	0.1885
114	N	14.0067	14.0067	-0.2407	-0.2407
115	H	1.0079	1.0079	0.1799	0.1799
116	N	14.0067	14.0067	-0.2514	-0.2514
117	H	1.0079	1.0079	0.1808	0.1808
118	H	1.0079	1.0079	0.1398	0.1398
119	H	1.0079	1.0079	0.1859	0.1859
120	H	1.0079	1.0079	0.1806	0.1806
121	N	14.0067	14.0067	-0.2653	-0.2653
122	N	14.0067	14.0067	-0.2409	-0.2409
123	Zn	65.3800	65.3800	0.5878	0.5878
124	C	12.0107	12.0107	-0.2539	-0.2539
125	C	12.0107	12.0107	-0.2133	-0.2133
126	C	12.0107	12.0107	0.3967	0.3967
127	C	12.0107	12.0107	-0.5821	-0.5821
128	C	12.0107	12.0107	-0.5855	-0.5855
129	N	14.0067	14.0067	-0.2430	-0.2430
130	H	1.0079	1.0079	0.1432	0.1432
131	H	1.0079	1.0079	0.1900	0.1900
132	H	1.0079	1.0079	0.1786	0.1786
133	C	12.0107	12.0107	-0.5742	-0.5742
134	H	1.0079	1.0079	0.1732	0.1732
135	H	1.0079	1.0079	0.1793	0.1793

136	N	14.0067	14.0067	-0.2517	-0.2517
137	C	12.0107	12.0107	-0.2220	-0.2220
138	H	1.0079	1.0079	0.1898	0.1898
139	H	1.0079	1.0079	0.1828	0.1828
140	C	12.0107	12.0107	0.3955	0.3955
141	H	1.0079	1.0079	0.1648	0.1648
142	N	14.0067	14.0067	-0.2515	-0.2515
143	H	1.0079	1.0079	0.1790	0.1790
144	C	12.0107	12.0107	-0.5840	-0.5840
145	N	14.0067	14.0067	-0.2575	-0.2575
146	Zn	65.3800	65.3800	0.5867	0.5867
147	C	12.0107	12.0107	-0.2241	-0.2241
148	C	12.0107	12.0107	-0.2013	-0.2013
149	H	1.0079	1.0079	0.1325	0.1325
150	Zn	65.3800	65.3800	0.5677	0.5677
151	N	14.0067	14.0067	-0.2720	-0.2720
152	C	12.0107	12.0107	-0.2235	-0.2235
153	H	1.0079	1.0079	0.1735	0.1735
154	N	14.0067	14.0067	-0.2565	-0.2565
155	H	1.0079	1.0079	0.1620	0.1620
156	N	14.0067	14.0067	-0.2461	-0.2461
157	C	12.0107	12.0107	-0.1919	-0.1919
158	H	1.0079	1.0079	0.1743	0.1743
159	N	14.0067	14.0067	-0.2513	-0.2513
160	H	1.0079	1.0079	0.1578	0.1578
161	H	1.0079	1.0079	0.1744	0.1744
162	H	1.0079	1.0079	0.1911	0.1911
163	H	1.0079	1.0079	0.1773	0.1773
164	Zn	65.3800	65.3800	0.6006	0.6006
165	C	12.0107	12.0107	-0.2346	-0.2346
166	C	12.0107	12.0107	0.3933	0.3933
167	C	12.0107	12.0107	0.4033	0.4033
168	N	14.0067	14.0067	-0.2378	-0.2378
169	C	12.0107	12.0107	-0.1959	-0.1959
170	H	1.0079	1.0079	0.1403	0.1403
171	C	12.0107	12.0107	-0.2548	-0.2548
172	C	12.0107	12.0107	-0.2231	-0.2231
173	Zn	65.3800	65.3800	0.5678	0.5678
174	C	12.0107	12.0107	-0.2223	-0.2223
175	H	1.0079	1.0079	0.1763	0.1763
176	N	14.0067	14.0067	-0.2579	-0.2579
177	H	1.0079	1.0079	0.1795	0.1795
178	C	12.0107	12.0107	-0.2161	-0.2161
179	N	14.0067	14.0067	-0.2415	-0.2415
180	C	12.0107	12.0107	-0.2141	-0.2141
181	H	1.0079	1.0079	0.1389	0.1389
182	C	12.0107	12.0107	-0.2382	-0.2382
183	H	1.0079	1.0079	0.1786	0.1786
184	H	1.0079	1.0079	0.1855	0.1855
185	N	14.0067	14.0067	-0.2608	-0.2608

186	H	1.0079	1.0079	0.1630	0.1630
187	H	1.0079	1.0079	0.1884	0.1884
188	C	12.0107	12.0107	-0.5781	-0.5781
189	H	1.0079	1.0079	0.1847	0.1847
190	N	14.0067	14.0067	-0.2484	-0.2484
191	N	14.0067	14.0067	-0.2679	-0.2679
192	C	12.0107	12.0107	-0.2393	-0.2393
193	H	1.0079	1.0079	0.1635	0.1635
194	H	1.0079	1.0079	0.1545	0.1545
195	H	1.0079	1.0079	0.1860	0.1860
196	H	1.0079	1.0079	0.1539	0.1539
197	C	12.0107	12.0107	-0.2465	-0.2465
198	C	12.0107	12.0107	0.4016	0.4016
199	N	14.0067	14.0067	-0.2690	-0.2690
200	C	12.0107	12.0107	0.3943	0.3943
201	N	14.0067	14.0067	-0.2494	-0.2494
202	N	14.0067	14.0067	-0.2544	-0.2544
203	C	12.0107	12.0107	-0.5589	-0.5589
204	C	12.0107	12.0107	-0.2130	-0.2130
205	Zn	65.3800	65.3800	0.5705	0.5705
206	C	12.0107	12.0107	0.4025	0.4025
207	H	1.0079	1.0079	0.1547	0.1547
208	C	12.0107	12.0107	-0.5874	-0.5874
209	N	14.0067	14.0067	-0.2529	-0.2529
210	C	12.0107	12.0107	-0.5874	-0.5874
211	C	12.0107	12.0107	-0.2137	-0.2137
212	C	12.0107	12.0107	-0.1959	-0.1959
213	H	1.0079	1.0079	0.1376	0.1376
214	C	12.0107	12.0107	-0.5775	-0.5775
215	H	1.0079	1.0079	0.1332	0.1332
216	H	1.0079	1.0079	0.1402	0.1402
217	H	1.0079	1.0079	0.1752	0.1752
218	H	1.0079	1.0079	0.1747	0.1747
219	H	1.0079	1.0079	0.1908	0.1908
220	N	14.0067	14.0067	-0.2650	-0.2650
221	H	1.0079	1.0079	0.1606	0.1606
222	H	1.0079	1.0079	0.1533	0.1533
223	C	12.0107	12.0107	-0.5814	-0.5814
224	H	1.0079	1.0079	0.1571	0.1571
225	C	12.0107	12.0107	-0.2275	-0.2275
226	N	14.0067	14.0067	-0.2582	-0.2582
227	H	1.0079	1.0079	0.1818	0.1818
228	C	12.0107	12.0107	-0.5691	-0.5691
229	H	1.0079	1.0079	0.1531	0.1531
230	H	1.0079	1.0079	0.1832	0.1832
231	C	12.0107	12.0107	-0.2304	-0.2304
232	C	12.0107	12.0107	-0.2367	-0.2367
233	C	12.0107	12.0107	-0.5639	-0.5639
234	C	12.0107	12.0107	-0.2007	-0.2007
235	H	1.0079	1.0079	0.1913	0.1913

236	C	12.0107	12.0107	-0.5564	-0.5564
237	H	1.0079	1.0079	0.1852	0.1852
238	N	14.0067	14.0067	-0.2498	-0.2498
239	C	12.0107	12.0107	-0.2236	-0.2236
240	H	1.0079	1.0079	0.1630	0.1630
241	H	1.0079	1.0079	0.1408	0.1408
242	H	1.0079	1.0079	0.1411	0.1411
243	N	14.0067	14.0067	-0.2457	-0.2457
244	H	1.0079	1.0079	0.1792	0.1792
245	C	12.0107	12.0107	-0.2033	-0.2033
246	C	12.0107	12.0107	-0.2484	-0.2484
247	H	1.0079	1.0079	0.1681	0.1681
248	H	1.0079	1.0079	0.1344	0.1344
249	C	12.0107	12.0107	0.3887	0.3887
250	C	12.0107	12.0107	-0.5664	-0.5664
251	C	12.0107	12.0107	-0.5599	-0.5599
252	C	12.0107	12.0107	-0.2054	-0.2054
253	C	12.0107	12.0107	-0.2162	-0.2162
254	H	1.0079	1.0079	0.1716	0.1716
255	H	1.0079	1.0079	0.1355	0.1355
256	C	12.0107	12.0107	0.4009	0.4009
257	H	1.0079	1.0079	0.1569	0.1569
258	H	1.0079	1.0079	0.1536	0.1536
259	C	12.0107	12.0107	-0.2192	-0.2192
260	N	14.0067	14.0067	-0.2577	-0.2577
261	C	12.0107	12.0107	0.4058	0.4058
262	H	1.0079	1.0079	0.1748	0.1748
263	H	1.0079	1.0079	0.1337	0.1337
264	C	12.0107	12.0107	-0.2437	-0.2437
265	N	14.0067	14.0067	-0.2607	-0.2607
266	C	12.0107	12.0107	0.4050	0.4050
267	C	12.0107	12.0107	0.3889	0.3889
268	N	14.0067	14.0067	-0.2415	-0.2415
269	N	14.0067	14.0067	-0.2633	-0.2633
270	H	1.0079	1.0079	0.1861	0.1861
271	C	12.0107	12.0107	-0.2538	-0.2538
272	H	1.0079	1.0079	0.1612	0.1612
273	H	1.0079	1.0079	0.1743	0.1743
274	H	1.0079	1.0079	0.1785	0.1785
275	H	1.0079	1.0079	0.1343	0.1343
276	H	1.0079	1.0079	0.1776	0.1776

Table 4.5 Field van der Waals parameters of atoms in a sodalite framework generated "by hand" (HAND) and automatically using the DOAP software (DOAP). The values are specifically identical showing that the DOAP software performs as expected in this capacity.

Atom # 1	Atom # 2	vdW Epsilon Value		vdW Sigma Value	
		HAND	DOAP	HAND	DOAP
Si	Si	0.40200	0.40200	3.82640	3.82640
Cx	Si	0.15463	0.15463	3.28570	3.28570
Cx	Cx	0.05948	0.05948	2.74500	2.74500
O	Si	0.15531	0.15531	3.47225	3.47225
O	Cx	0.05974	0.05974	2.93155	2.93155
O	O	0.06000	0.06000	3.11810	3.11810
Ox	Si	0.26160	0.26160	3.42170	3.42170
Ox	Cx	0.10062	0.10062	2.88100	2.88100
Ox	O	0.10106	0.10106	3.06755	3.06755
Ox	Ox	0.17023	0.17023	3.01700	3.01700

Table 4.6 Field van der Waals parameters in a ZIF 8 framework generated "by hand" (HAND) and automatically using the DOAP software (DOAP). The values are specifically identical showing that the DOAP software performs as expected in this capacity.

Atom # 1	Atom # 2	vdW Epsilon Value		vdW Sigma Value	
		HAND	DOAP	HAND	DOAP
Zn	Zn	0.12400	0.12400	2.46160	2.46160
H	Zn	0.07386	0.07386	2.51635	2.51635
H	H	0.04400	0.04400	2.57110	2.57110
C	Zn	0.11411	0.11411	2.94625	2.94625
C	H	0.06797	0.06797	3.00100	3.00100
C	C	0.10500	0.10500	3.43090	3.43090
N	Zn	0.09250	0.09250	2.86115	2.86115
N	H	0.05510	0.05510	2.91590	2.91590
N	C	0.08512	0.08512	3.34580	3.34580
N	N	0.06900	0.06900	3.26070	3.26070
Cx	Zn	0.08588	0.08588	2.60330	2.60330
Cx	H	0.05116	0.05116	2.65805	2.65805
Cx	C	0.07903	0.07903	3.08795	3.08795
Cx	N	0.06406	0.06406	3.00285	3.00285
Cx	Cx	0.05948	0.05948	2.74500	2.74500
Ox	Zn	0.14529	0.14529	2.73930	2.73930
Ox	H	0.08655	0.08655	2.79405	2.79405
Ox	C	0.13369	0.13369	3.22395	3.22395
Ox	N	0.10838	0.10838	3.13885	3.13885
Ox	Cx	0.10062	0.10062	2.88100	2.88100
Ox	Ox	0.17023	0.17023	3.01700	3.01700

As one might expect, due to the causal nature of the calculations, the charges and field parameters in tables 4.3 – 4.6 do not differ at all. Alternatively, due to the random sampling employed by the GCMC method, the gas uptake values, whose comparison between an automated and non-automated approach shown below in figures 4.2 and 4.3 differ slightly, but are qualitatively consistent with one another. This suggests that the DOAP automation software is adequately performing the simulation as though it were performed on a manual basis.

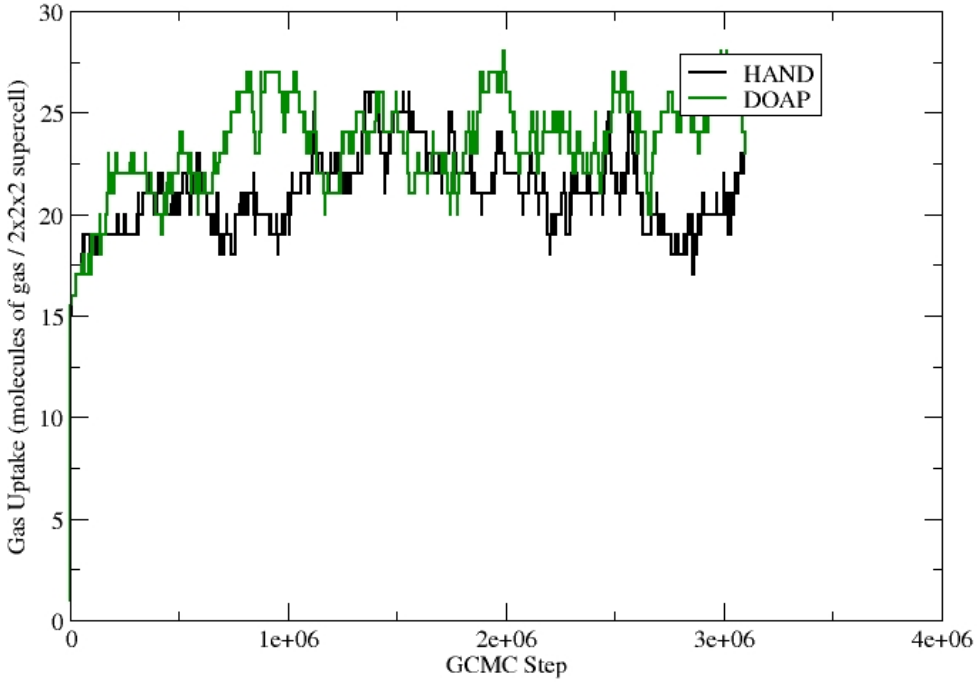


Figure 4.2 Overlay of "by hand" (black) and DOAP automated (green) gas uptake simulation results for the adsorption of carbon dioxide in sodalite.

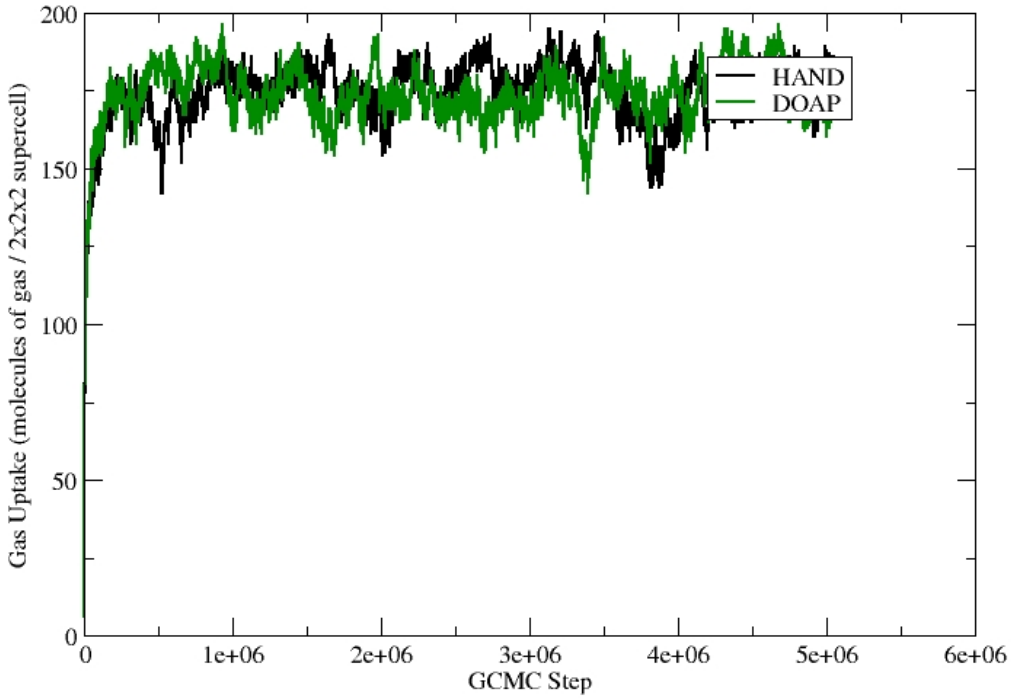


Figure 4.3 Overlay of "by hand" (black) and DOAP automated (green) gas uptake simulation results for the adsorption of carbon dioxide in the ZIF 8 framework.

In general from comparison the intermediate simulation inputs through tables 4.1 – 4.6 and figures 4.2 – 4.3 it is expected that the nature of the simulations performed using the DOAP software will reflect the same results as though performed by hand in a manual fashion.

Chapter 5 - Assessment of Convergence

5.1 - Intent of convergence assessment

The success of any simulation not only depends on whether the theory underlying the simulation is valid. It is equally important to make sure that that theory is implemented and carried out correctly. An important aspect of carrying out a proper simulation is ensuring that the simulation is performed to completion. A simulation is generally considered to be complete when the property one wishes to determine from the simulation remains relatively constant over an extended period of time, such that we can assert that should the simulation continue, we would no longer except the value being observed to change significantly. When a simulation reaches such a state, it is said to be converged. The measurement of convergence depends on the nature of the observable being sought through the simulation. A system that oscillates between two states periodically may take on a different criteria for convergence than one with a single state. Likewise, systems with a great amount of fluctuation will require assessments based on different metrics to ensure that the system is well behaved. In the case of the DOAP project, finding a convergence scheme that depicts when a framework is done adsorbing gas molecules is needed in order to complete the automation of the process.

5.2 - Convergence Criteria Risk Assessment

The proper assessment of convergence carries with it an associated level of risk. Ideally we would like a simulation to be stopped as soon as it is converged. Should the simulation not run long enough, the risk that the observable is not properly represented increases. Should the simulation run for too long, computing resources are wasted needlessly and part of the intent of automating a gas uptake simulation is lost. The risk associated with running a simulation for too long is directly proportional to the rate of progression through a simulation. This is dependent on the efficiency of the software package used.

Throughout the development of the DOAP software, two GCMC software packages have been used. The first, the DL_POLY GCMC code written in Python by Andrew Sirjoosingh, encapsulated the DL_POLY molecular mechanics program to run single point energy calculations. Performing a GCMC simulation in this manner proves to be inefficient, because of the amount of processing overhead associated with using DL_POLY to run a single feature it contains. With this in mind, the second GCMC software package used by the DOAP software, the *fastmc* code written in Fortran by Peter Boyd is coded specifically to perform GCMC calculations only. The selection of programming languages and specificity for GCMC calculations is reflected in the speed at which the two software packages perform the simulations.

Some preliminary comparisons were done to assess the increase in speed by using the *fastmc* package. Both systems were used to evaluate 1 million GCMC steps of the ZIF-8 framework, a 276 atom system, with a starting occupancy of zero carbon

dioxide molecules. The DL_POLY based GCMC code log revealed that the simulation took approximately 5800 minutes. The same simulation using the *fastmc* GCMC simulation software took approximately 189 minutes to perform the same calculation. The increase in the rate of acquisition is approximate thirty one times by using the faster *fastmc* software package. This speed increase is reflected in the risk associated with performing an overly long simulation. Should the DOAP software suggest that the simulation need to terminated at a point past the point of actual convergence, the associated time in processing the fruitless simulation steps will occur much faster than when initially ran in the DL_POLY code. This directly impacts the level of certainty that we expect the convergence assessment to have, as it become generally more affordable to wager the processing of potentially needless GCMC steps against the chance of an increased likelihood that the simulation is properly converged.

5.3 - Assessment of Convergence Layout

In order to build a representative set of GCMC simulation data to generate convergence assessment metrics for, a program was written to scour the Wooki cluster file system for GCMC simulation output files. The program identified data directories by looking for “branchXX” directories containing numguests.out files, the output files of the DL_POLY GCMC code previously mentioned. Branch directories are made because multiple branches of the GCMC simulation were run in parallel for the GCMC code. Upon finding a GCMC data set, the program would copy, sort and distinctly label the directory containing the output files from the simulation. The file crawler found over

300 simulation file sets. These initial sets were reduced to 28 expected to account for most of the variability in the characteristics of the uptake graphs found. Sets were selected against when showing abnormal termination, missing branches, low number of branches, duplication of results with other sets, subsets of other sets and otherwise characteristically speculative results.

Each of the remaining 28 sets, containing up to eight GCMC branches each were averaged across branches step-wise into one “average branch” file representing the average uptake amongst all branches at each step. All metrics to characterize the convergence of the system were based on this average branch file. Figures 5.1, 5.2, 5.3 depict the gas uptake data vs the GCMC step for each of the 28 test set outputs below.

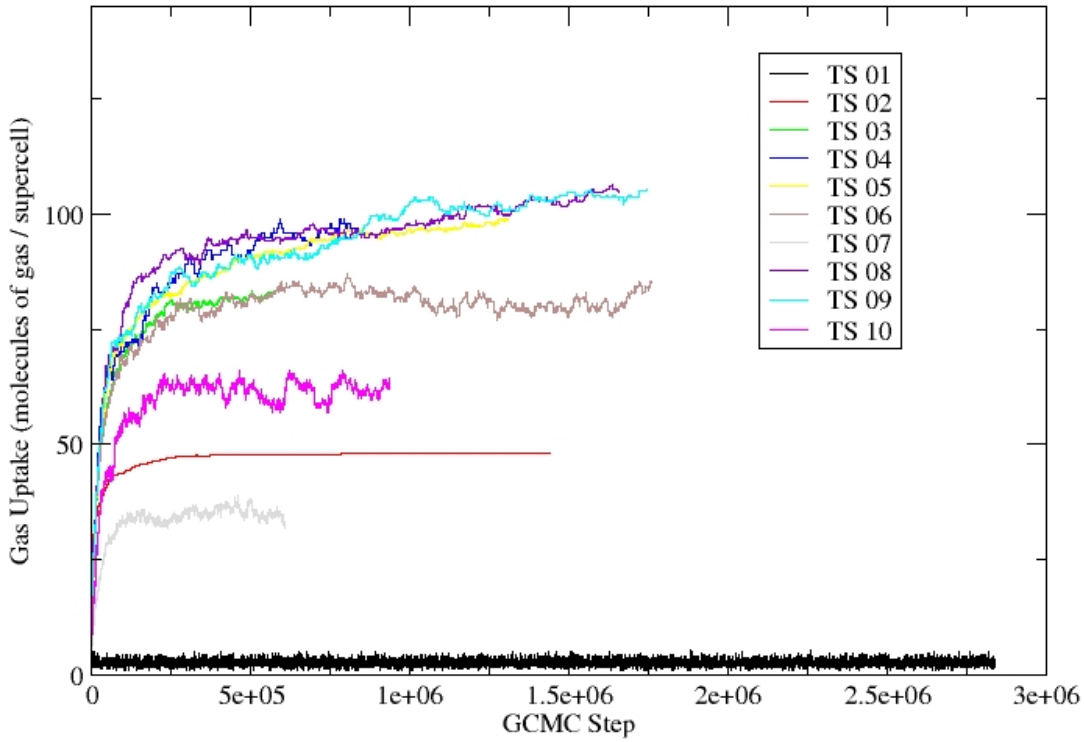


Figure 5.1 Depiction of gas uptake data for test sets (TS) 01 – 10.

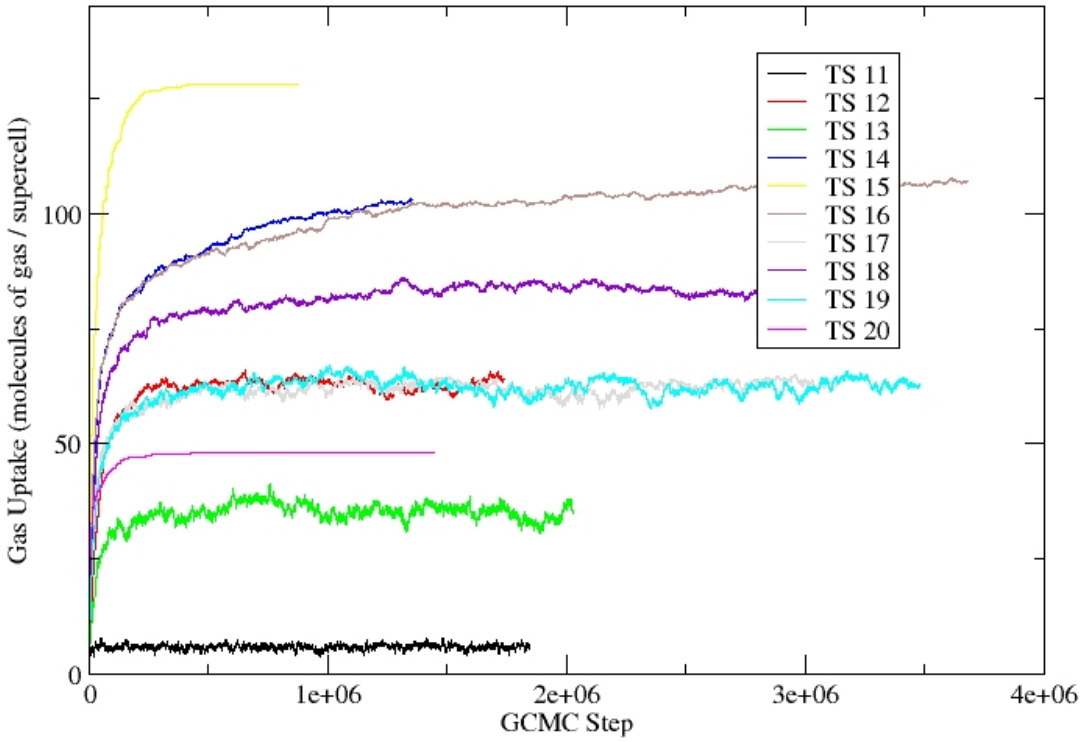


Figure 5.2 Depiction of gas uptake data for test sets (TS) 11 – 20.

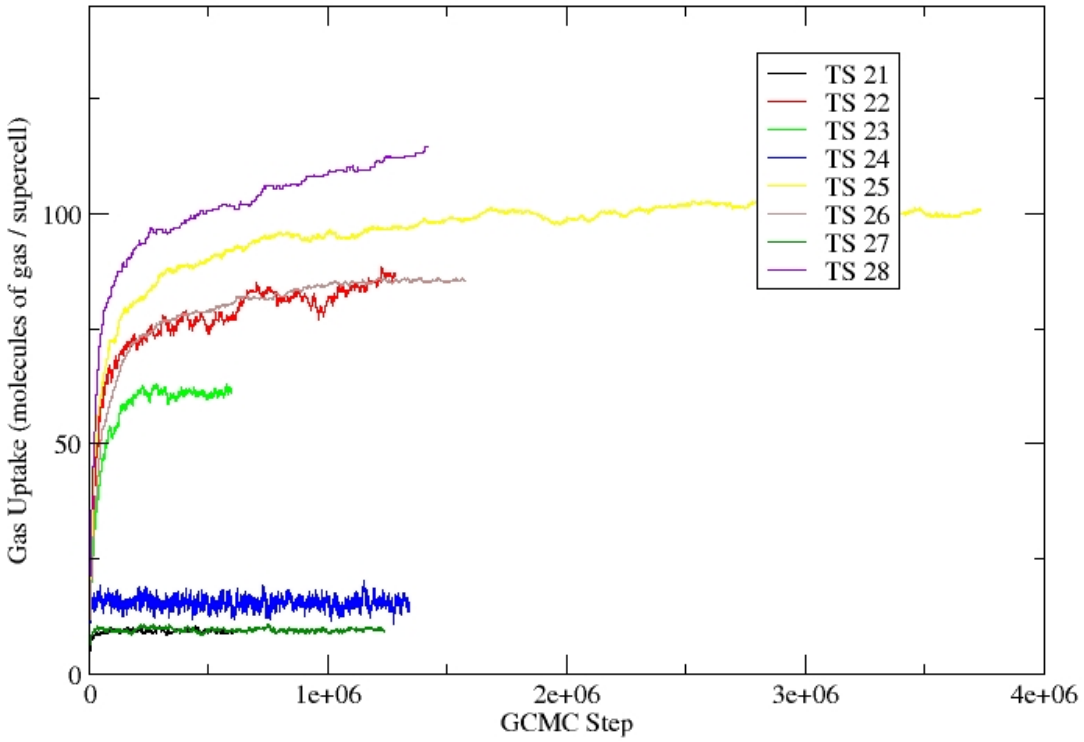


Figure 5.3 Depiction of gas uptake data for test sets (TS) 21 – 28.

Table 5.1 below presents the point of perceived convergence for the gas uptake data from each of the 28 test sets.

Table 5.1 Perceived points of convergence for Test Sets 01 – 28 used in the convergence metric assessment. Those indicating “Does Not Converge” were test sets where no convergence point was reach and were used as negative controls in evaluating the metrics.

Test Set	Point of Convergence (GCMC Step)
01	35000
02	500000
03	Does Not Converge
04	Does Not Converge
05	Does Not Converge
06	750000
07	400000
08	Does Not Converge
09	Does Not Converge
10	300000
11	400000
12	650000
13	550000
14	Does Not Converge
15	500000
16	3000000
17	1000000
18	1500000
19	1400000
20	480000
21	150000
22	Does Not Converge
23	275000
24	75000
25	2300000
26	1200000
27	275000
28	Does Not Converge

5.4 - Discussion of Metrics

Five metrics were investigated with the intent of characterizing the converged state of the simulation. They are:

1 – The slope of a windowed line of best fit

2 – The standard deviation of data within a windowed range

3 – The absolute domain of data within a windowed range

4 – The maximum deviation from the mean within a windowed range

5 – The frequency of new maximum values along successive points

For metrics 1 - 4, each of the metrics were evaluated for window sizes varying from 100000 – 950000 data points in 5000 point increments at a 50000 step interval. This resulted in the generation of approximately 200 plots per test system, per metric, comprising over 195 GB of data, of which each graph was evaluated by hand to determine the window size and threshold value which was most representative of a predetermined point of convergence. In all test set cases for metrics 1 through 4, no single threshold value or window size was universally applicable. Due to the extreme variability of parameters between sets, it was not feasible to test for a specific threshold or window size. Selecting for a threshold value that would adequately capture the nature of the convergence in one test set would result in the remaining sets being completely converged very early or never converged at any point. In some cases the threshold of convergence crossed in one test set was below the level that another experienced throughout the entire simulation. In other cases, that threshold was passed very quickly and provided no information about the convergence point.

When assessing each test set against each metric by hand, cases were sought whereby the metric in question trended towards an indication of convergence in a consistent fashion. That is to say that in the event that two cases existed, one where a threshold was crossed at or near the expected point of convergence and then continued in the same fashion and one where the threshold was crossed, but erratically changed directions or did not have consistent behaviour, the latter was discarded.

Slope of a windowed line of best fit:

In a preliminary approach to characterize the convergence of the system, the slope was taken of a line of best fit for a window of a number of data points. This was done using a minimized root mean square distance fit, where the slope of the resulting line can be expressed as follows:

$$slope = \frac{n\sum xy - (\sum x)(\sum y)}{n\sum (x^2) - (\sum x)^2} \quad \textbf{(5.1)}$$

Equation 5.1 is a ratio of the covariance of x and y over the variance of x and describes the slope of line which intersects the points of the plot, whereby square root of the sum of the squares of the distances between the points and the line defined with the indicated slope are minimized.

No consistent trend was found between sample simulation sets. The size of the preferential window needed to provide a definitive convergence point differed between

sets. Limits for the magnitude of the slope needed to be surpassed to confer convergence varied between sets inconsistently as well.

One such set, test set 18, which was able to indicate the state of its convergence, which fell at 1500000 GCMC steps, baring a slope threshold of 2.45E-06 was reached. It is depicted in figure 5.5, which shows the slope of line of best fit using a 900000 step window at 5000 step intervals for the gas uptake data presented in figure 5.4 below. It is also of note that in figure 5.5, the slope of line of best fit stops to describe the convergence beyond the point of convergence, that is, it rises above the threshold value, which is counter indicative to what we expect, given that once the system has reached convergence, by definition it should remain converged.

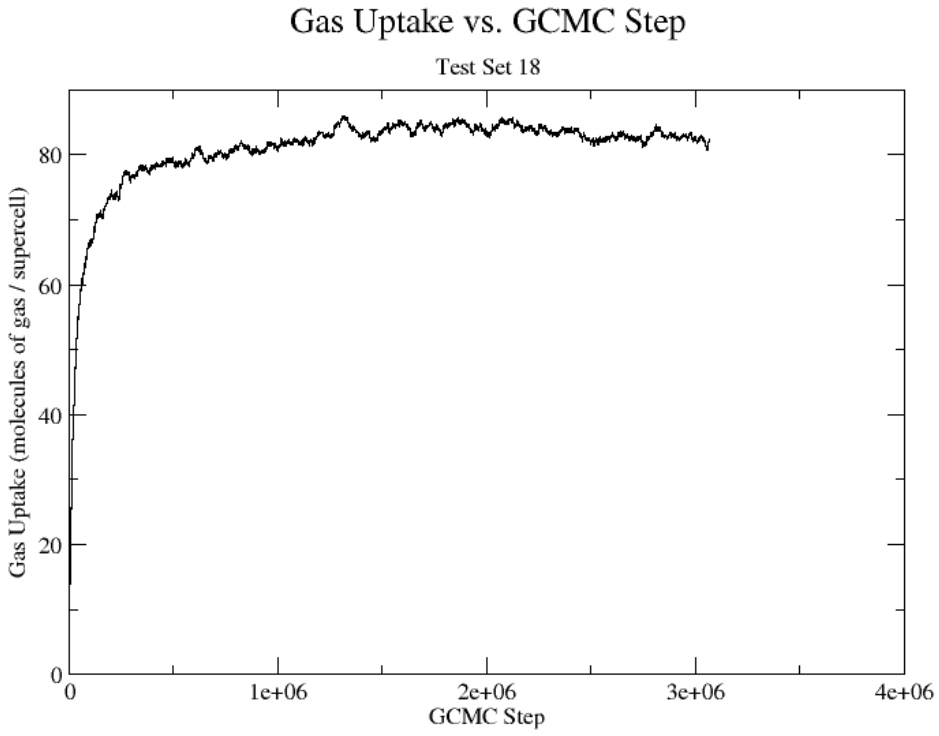


Figure 5.4 Plot of gas uptake for test set 1, showing molecules of gas adsorbed by the MOF supercell throughout the progression of the GCMC simulation steps.

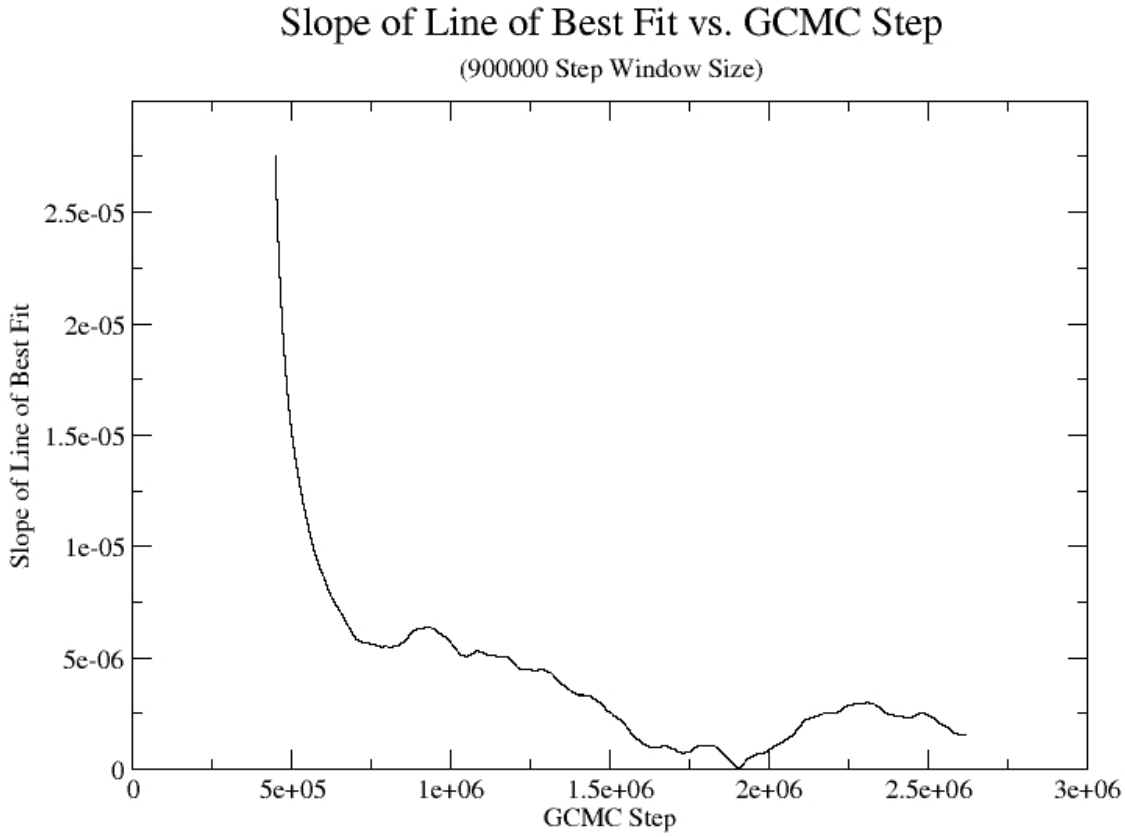


Figure 5.5 Plot of the slope of line of best fit for the gas uptake in the simulation of test set 18 over the course of the GCMC simulation steps. The slopes are generated for lines fit to windows of 900000 steps evaluated at intervals of 5000 steps.

Seven of the twenty eight sample sets could not have their convergence described even loosely using the slope of the line of best fit as an indicator regardless of the window size and threshold value used. Figure 5.7 depicts one such occurrence wherein a scan of all window sizes did not describe the nature of the convergence of test set 11 whose uptake progression throughout the GCMC simulation is depicted in

figure 5.6. It is evident that there is no ascertainable threshold within the plot of figure 5.7 which will indicate the point of convergence observed at 400000 GCMC step.

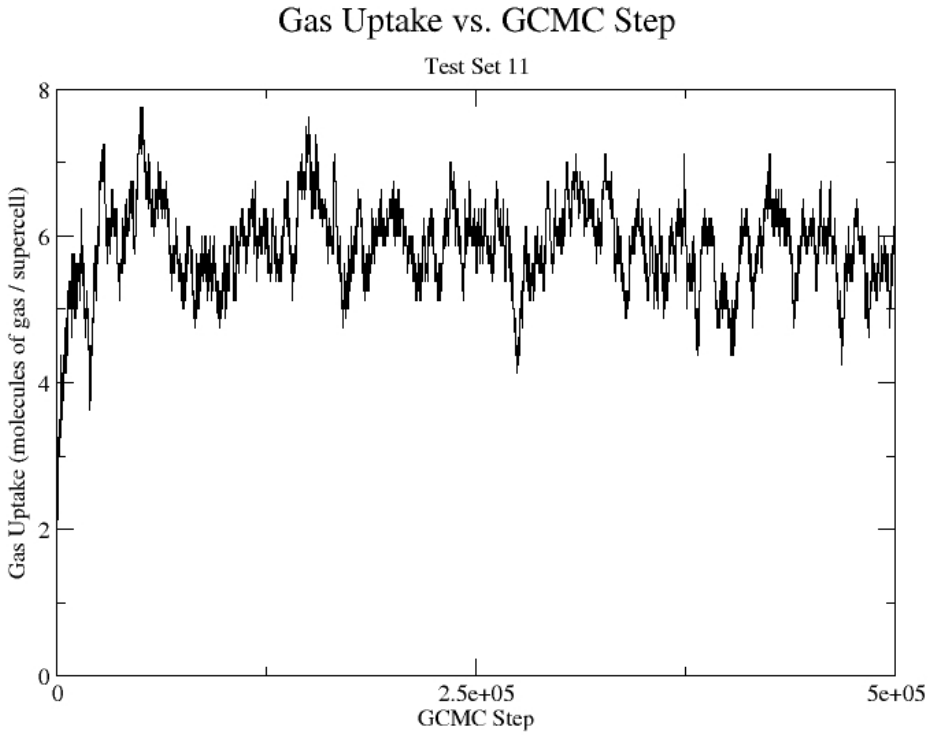


Figure 5.6 Plot of gas uptake for test set 11, showing molecules of gas adsorbed by the MOF supercell throughout the progression of the GCMC simulation steps.

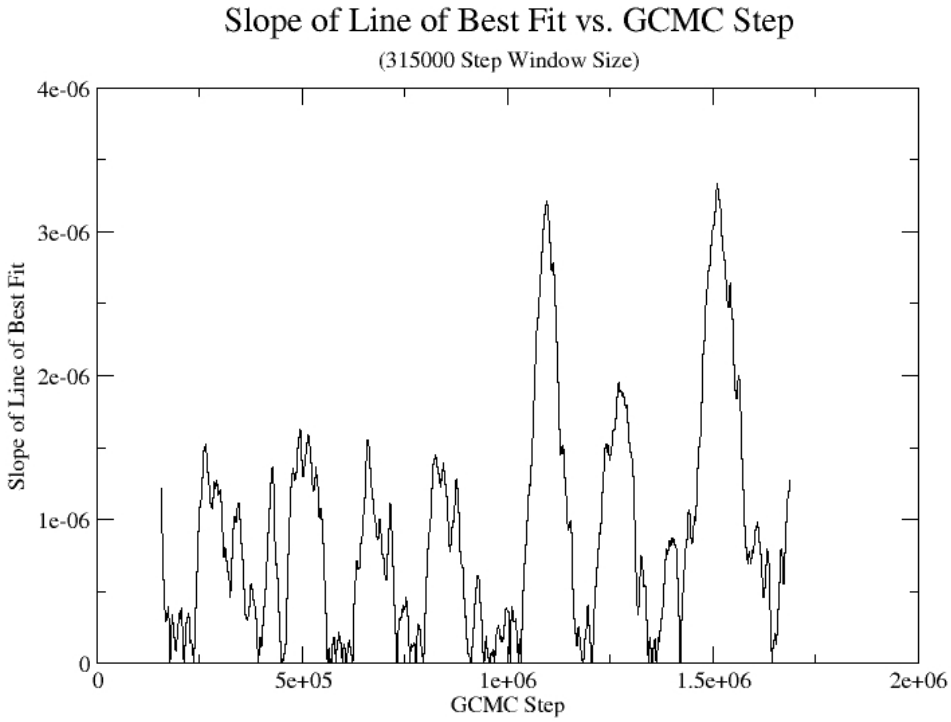


Figure 5.7 Plot of the slope of line of best fit for the gas uptake in the simulation of test set 11 over the course of the GCMC simulation steps. The slopes are generated for lines fit to windows of 315000 steps evaluated at intervals of 5000 steps.

Of the remaining sample sets, the slope tended to follow the convergence beyond the point at which it indicated convergence for fourteen of the sets. The seven remaining samples were able to reach threshold values at or shortly beyond the proposed convergence point, however, fluctuated above and below the threshold value beyond that point without ascertainable cause. Of the 8 sets which were not converged and used as negative controls, all but one showed a consistent trend downward, which one might expect to possibly reach a level indicating convergence

should the simulation continue. Figure 5.9 show the slope of line of best fit for 300000 step windows of the uptake data presented in figure 5.8. The slope trend continues to indicate a converging uptake although no threshold is reached by which complete convergence of the system can be indicated. It is likely that should the simulation had continue to run, one might expect that the slope trend would continue to approach a threshold value, which would indicate the convergence of the system.

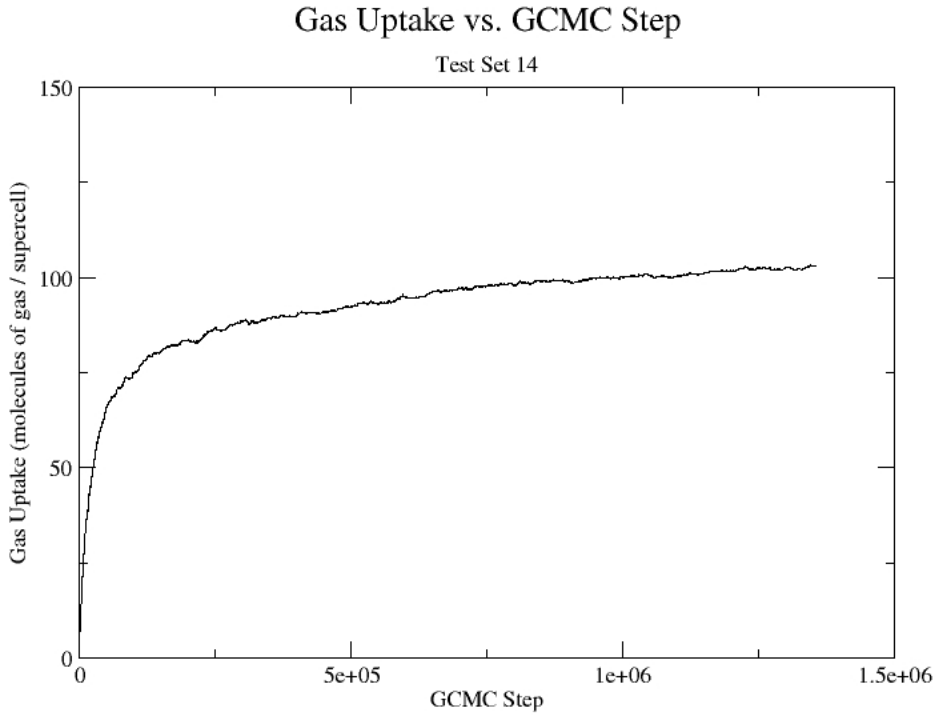


Figure 5.8 Plot of gas uptake for test set 14, showing molecules of gas adsorbed by the MOF supercell throughout the progression of the GCMC simulation steps.

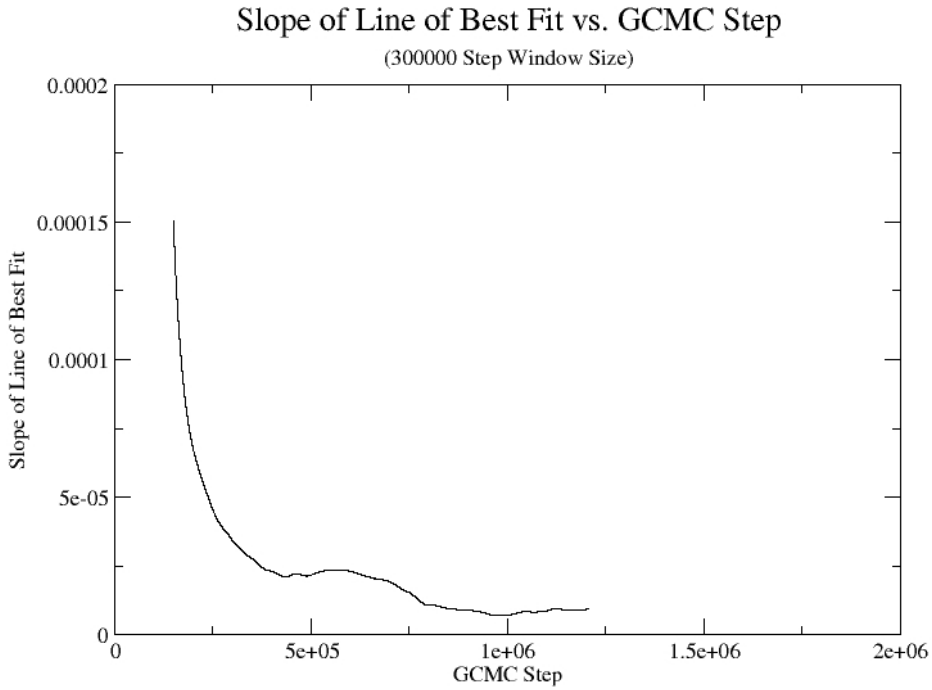


Figure 5.9 Plot of the slope of line of best fit for the gas uptake in the simulation of test set 14 over the course of the GCMC simulation steps. The slopes are generated for lines fit to windows of 300000 steps evaluated at intervals of 5000 steps.

Overall, the information observed suggests that the slope of line of best fit is not a good indicator of convergence in this application. It was found that the slope of line of best fit could not account for frameworks having low uptakes, and likewise affected by the amount of fluctuation contained within the uptake value. Having larger fluctuations within the data required a larger window to attenuate the influence of the varying data. The parameter values used to describe convergence using the slope of

the line of best fit method is shown in table 5.2 below. The logical progression from the metric was to provide a measure of the dispersed nature of the values of some simulations.

Table 5.2 Characterization of convergence assessment parameters for slope of line of best fit convergence metric tested on test sets 01-28.

Sample ID	Best window size	Metric threshold value	Trend continues?
01	No trend	No trend	N/A
02	800000	2.79E-006	Yes
03	350000	Not converged	Yes
04	300000	Not converged	Yes
05	550000	Not converged	Yes
06	600000	7.98E-007	No
07	285000	3.73E-006	Yes
08	Not converged	Not converged	No
09	850000	Not converged	Yes
10	350000	1.37E-005	No
11	No trend	No trend	N/A
12	900000	1.80E-006	No
13	700000	5.04E-006	No
14	300000	Not converged	Yes
15	200000	3.15E-007	Yes
16	900000	1.76E-006	Yes
17	No trend	No trend	N/A
18	900000	2.45E-006	No
19	No trend	No trend	N/A
29	700000	1.54E-006	Yes
21	100000	4.20E-008	No
22	700000	Not converged	Yes
23	300000	7.91E-006	Yes
24	No trend	No trend	N/A
25	No trend	No trend	N/A
26	500000	2.40E-006	Yes
27	No trend	No trend	N/A
28	600000	Not converged	Yes

The standard deviation of data within a windowed range

The standard deviation was taken for each of the 28 samples, varying the window size. Based on its performance this metric predicted convergence comparatively better than the slope of line of best fit.

Five of the samples were unable to have their convergence point described by the standard deviation.

Of those remaining that were adequately indicated by the standard deviation, fifteen samples were observed to reach limits that continued to indicate convergence beyond the suggested point of convergence.

The resulting information for one such set is depicted in figure 5.11 below which depicts the windowed standard deviation in 300000 step windows for the uptake data in figure 5.10 pertaining to test set 2. It can be seen that the convergence point roughly attained at 500000 steps in figure 5.10 can be described by the threshold value of 9E-02 in figure 5.11 and that reasonably, the trend continues to show convergence beyond that point.

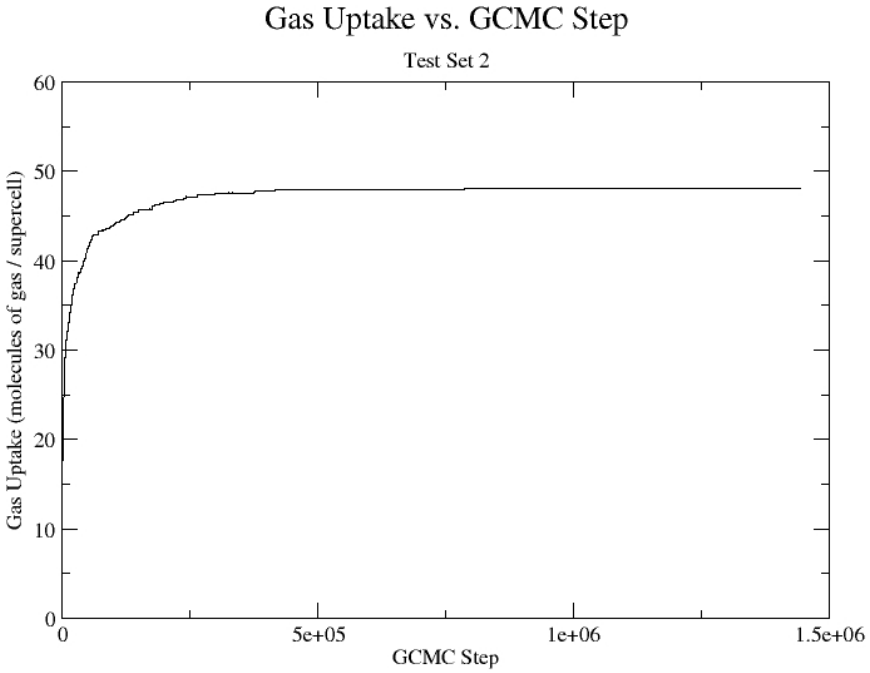


Figure 5.10 Plot of gas uptake for test set 2, showing molecules of gas adsorbed by the MOF supercell throughout the progression of the GCMC simulation steps.

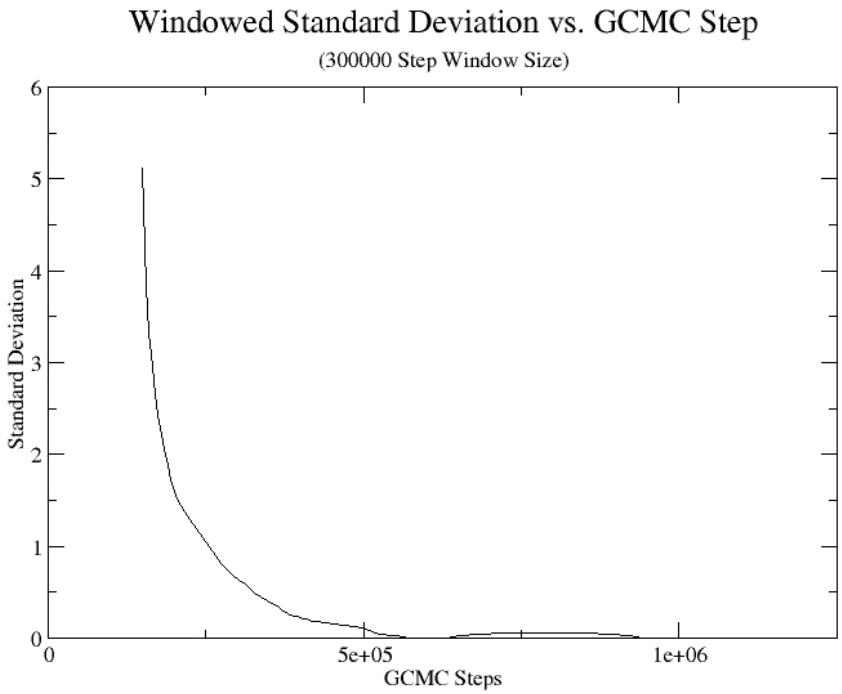


Figure 5.11 Plot of the windowed standard deviation of the gas uptake in the simulation of test set 2 over the course of the GCMC simulation steps. The standard deviations are generated for windows of 300000 steps evaluated at intervals of 5000 steps.

All but one of the non-converged negative control sets showed a consistent trend downwards. This can be seen in example by test set 5 depicted in figure 5.12 below whereby the slope associated with the uptake presented in figure 5.12 can be seen in figure 5.13 and shows a consistent downward trend towards convergence, although convergence in figure 5.12 is never reached. One would expect that should this trend continue a threshold value would be ascertainable by which the state of convergence would be described.

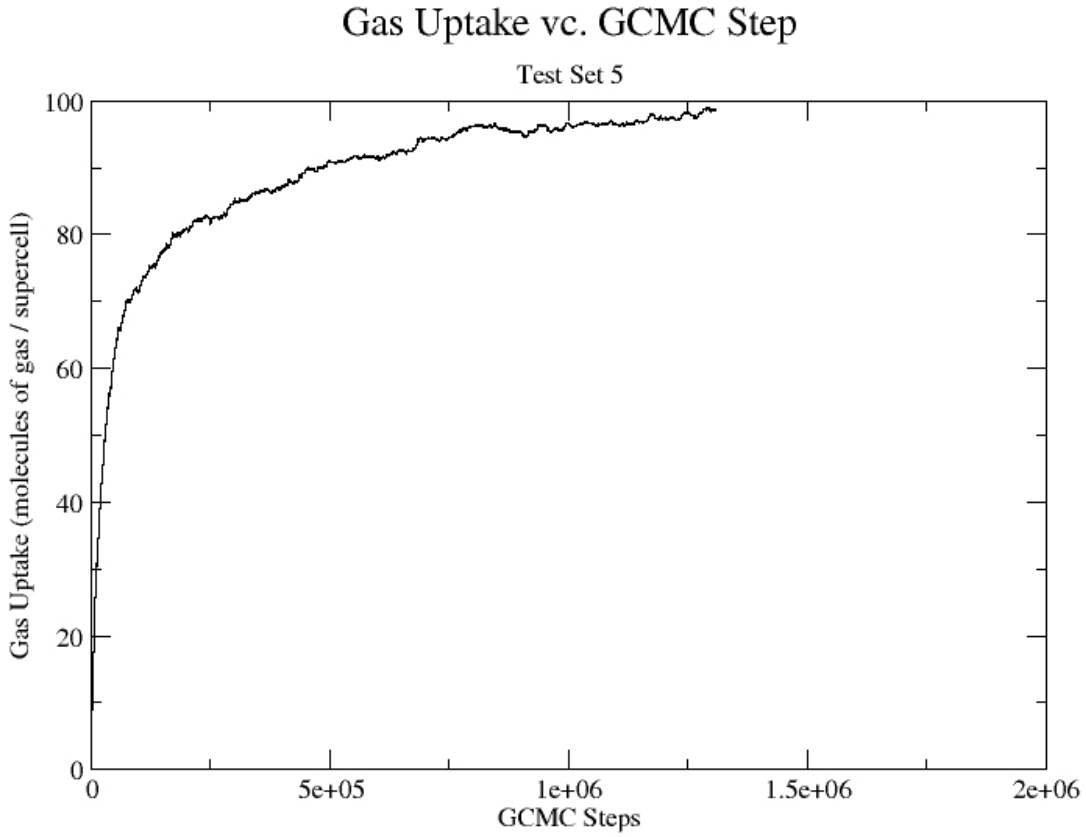


Figure 5.12 Plot of gas uptake for test set 5, showing molecules of gas adsorbed by the MOF supercell throughout the progression of the GCMC simulation steps.

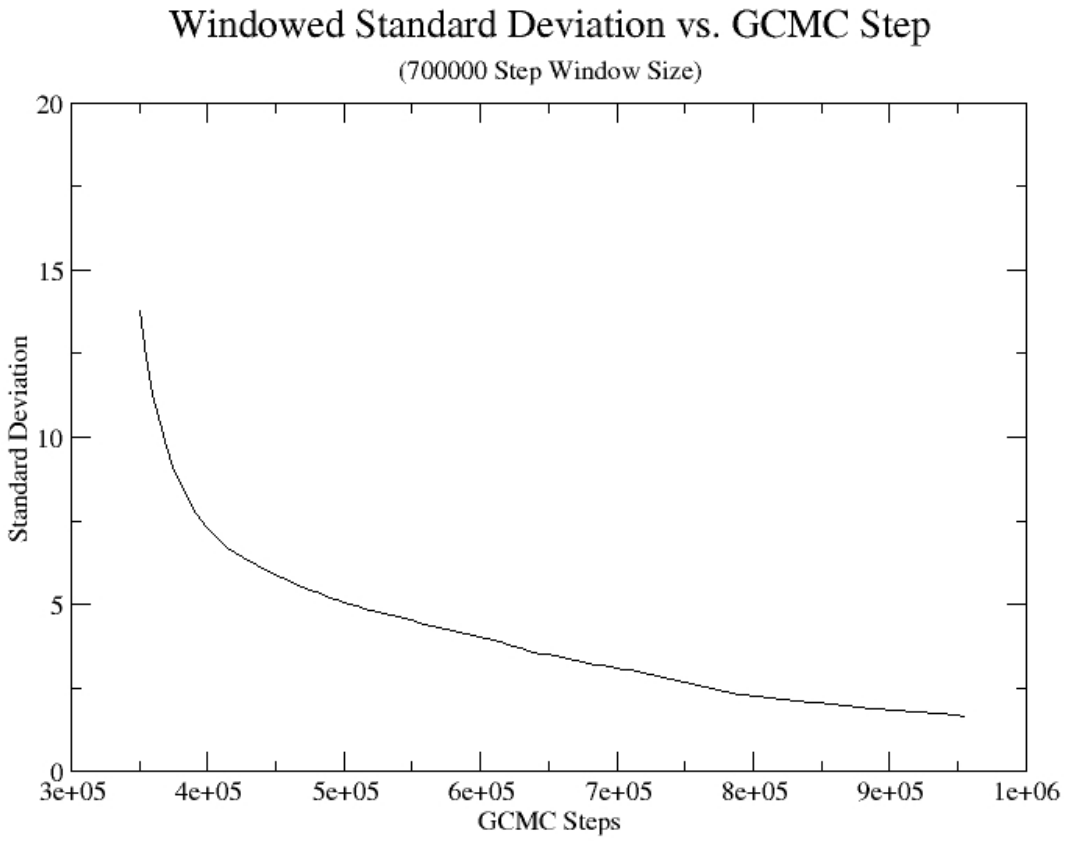


Figure 5.13 Plot of the windowed standard deviation of the gas uptake in the simulation of test set 5 over the course of the GCMC simulation steps. The standard deviations are generated for windows of 700000 steps evaluated at intervals of 5000 steps.

Overall, the indicator values of the standard deviation as well as the preferred window size varied drastically as before for each sample taken. It is also of note that the standard deviation calculation is resource intensive in Python, and took a relatively large amount of time to calculate compared to other metrics when many sample data

points were involved. While the standard deviation appears to perform better as an indicator of convergence than the slope of line of best fit, its demand of computing resources makes it ultimate utility in a real-time assessment questionable. This stems from the need to evaluate an average for the deviations from the means for each window, in each case. In effect, this metric is a composition of two averages. The first, used to measure the distance from the mean for each point, the second, an average of the distances. While there are mathematical simplifications that can be used to facilitate the calculation of the first average to avoid redundant calculations, the second average cannot be simplified in such a fashion and must be evaluated explicitly for all points, resulting in a relatively large amount of computational overhead compared to the evaluation of the other metrics in this study. The parameter values used to describe convergence using the windowed standard method is shown in table 5.3 below.

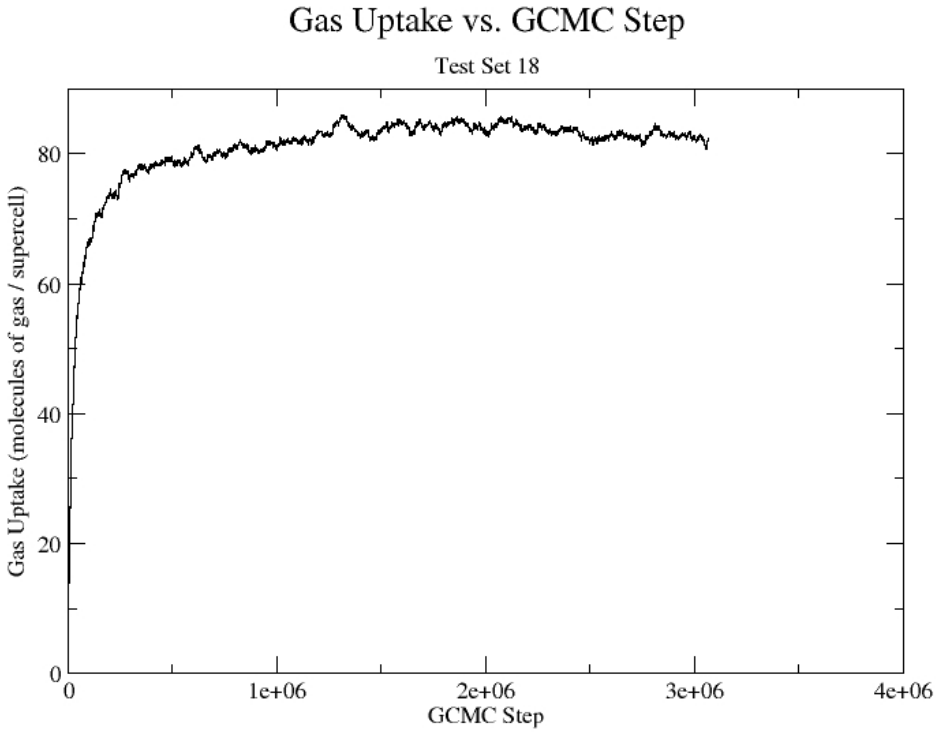
Table 5.3 Characterization of convergence assessment parameters for windowed standard deviation convergence metric tested on test sets 01-28.

Sample ID	Best window size	Metric threshold value	Trend continues?
01	No trend	No trend	No
02	300000	9.00E-002	Yes
03	300000	Not converged	Yes
04	350000	Not converged	Yes
05	700000	Not converged	Yes
06	900000	1.91	No
07	385000	1.24E+000	Yes
08	Not converged	Not converged	No
09	900000	Not converged	Yes
10	300000	1.84E+000	No
11	600000	0.517	No
12	900000	0.938	No
13	700000	1.79	No
14	500000	Not converged	Yes
15	500000	0.44	Yes
16	950000	0.79	Yes
17	No trend	No trend	N/A
18	900000	1.03	Yes
19	No trend	No trend	N/A
29	500000	0.18	Yes
21	200000	0.29	No
22	650000	Not converged	Yes
23	500000	5.76	Yes
24	No trend	No Trend	N/A
25	No trend	No Trend	N/A
26	700000	0.58	Yes
27	100000	0.22	No
28	700000	Not converged	Yes

The absolute domain of data within a windowed range

In an attempt to provide a comparable but cheaper metric than the standard deviation, the domain of windowed ranges of data are calculated. Requiring fewer calculations overall, this metric can be provided with much greater ease, however, overall the domain seems to be less of a descriptor of convergence than the standard deviation. Three of the twenty eight samples were unable to have their convergence

point described by the standard deviation, however, of those set which were describable, eighteen followed a consistent trend throughout the measurement. One such sample, test set 18, whose domain is described in figure 5.15 below shows a consistent towards a threshold of 5.6 using a window size of 900000 steps at which point the indication of convergence at approximately 1500000 steps is indicated.



+

Figure 5.14 Plot of gas uptake for test set 18, showing molecules of gas adsorbed by the MOF supercell throughout the progression of the GCMC simulation steps.

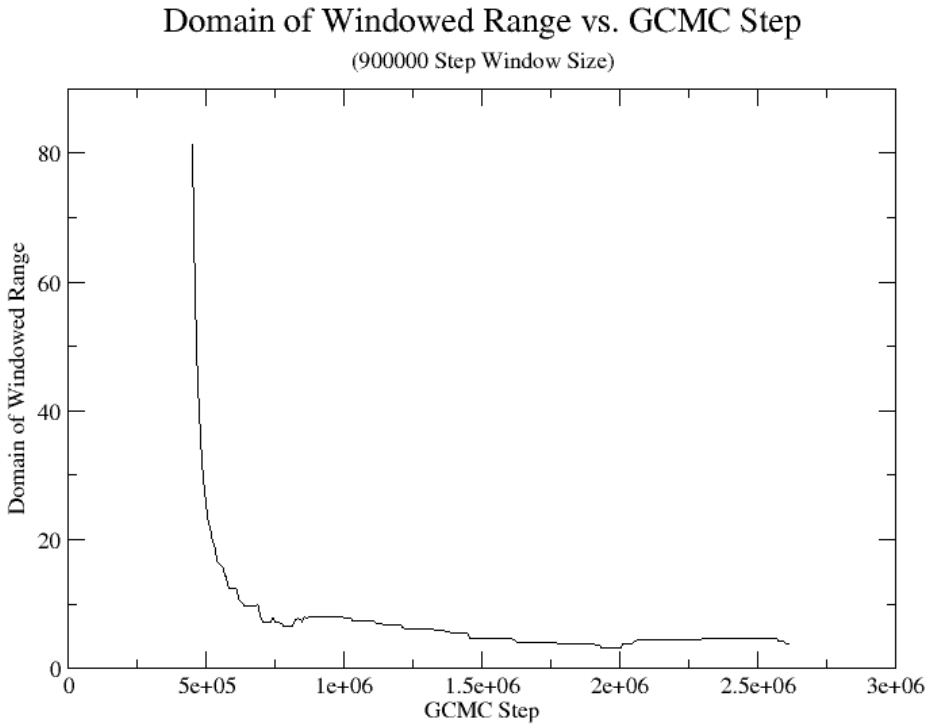


Figure 5.15 Plot of the domain of the windowed range for the gas uptake in the simulation of test set 18 over the course of the GCMC simulation steps. The domains are generated for windows of 900000 steps evaluated at intervals of 5000 steps.

As in all previous metrics, the non-convergence of the system was adequately represented such that a downwards trend could be established for many of the non-converged negative control test sets. One such test set, test set 4, is depicted in figure 5.17, showing a trend of the domain decreasing as the system approaches convergence.

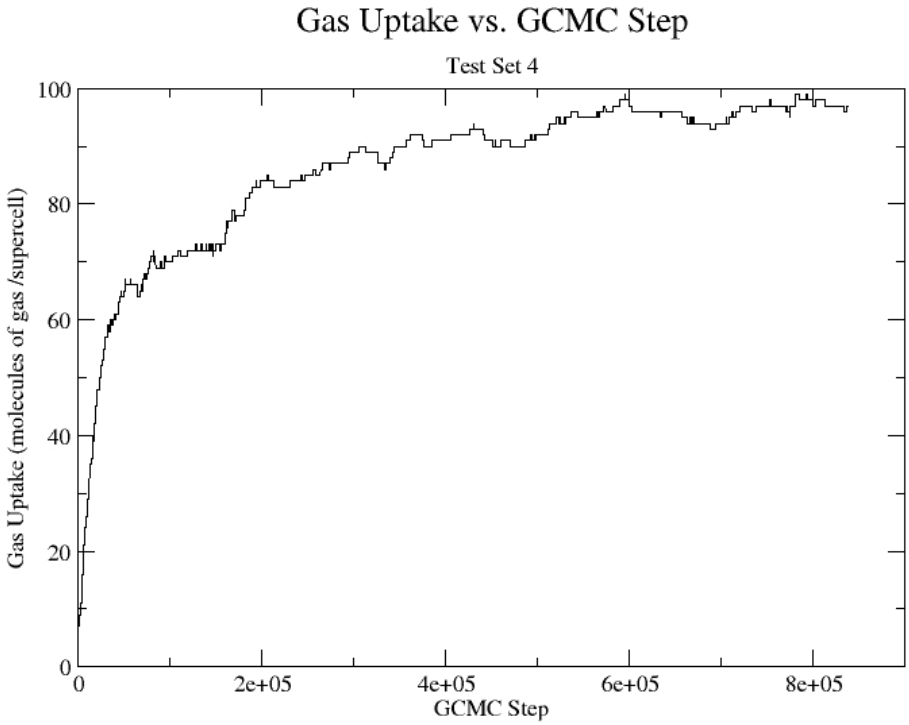


Figure 5.16 Plot of gas uptake for test set 4, showing molecules of gas adsorbed by the MOF supercell throughout the progression of the GCMC simulation steps.

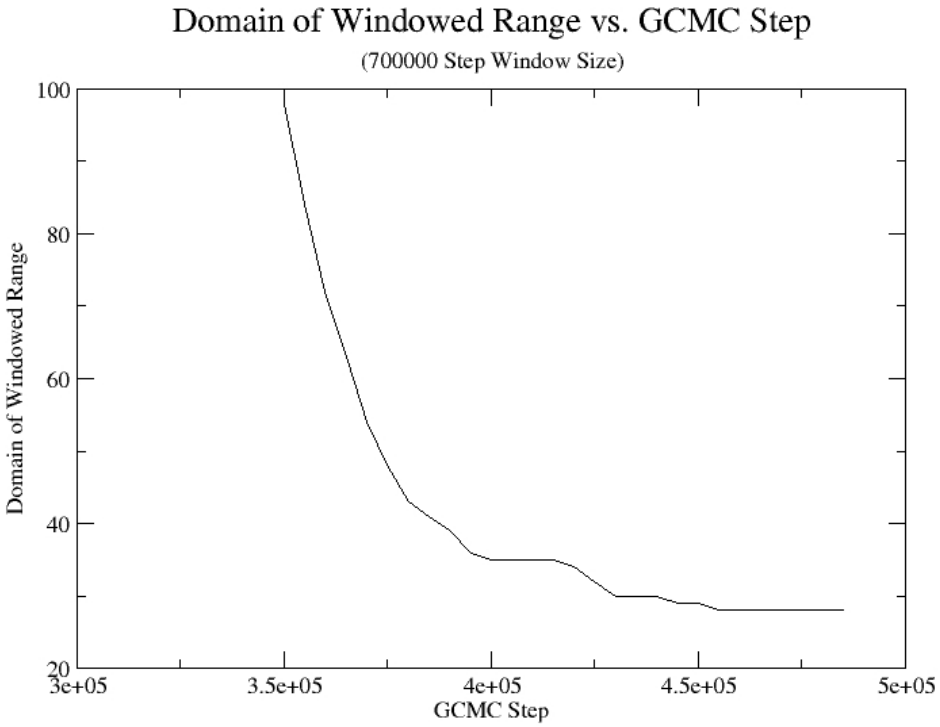


Figure 5.17 Plot of the domain of the windowed range for the gas uptake in the simulation of test set 4 over the course of the GCMC simulation steps. The domains are generated for windows of 700000 steps evaluated at intervals of 5000 steps.

Overall, the preferred window size, as before, did not follow any specific trend and the threshold value depicting the point of convergence varied widely as before. That is to say, there was no single set of values that could be used to express the convergence using the domain of a windowed range. The parameter values used to describe convergence using the windowed domain method is shown in table 5.4 below.

Table 5.4 Characterization of convergence assessment parameters for windowed domain convergence metric tested on test sets 01-28.

Sample ID	Best window size	Metric threshold value	Trend continues?
01	No trend	No trend	N/A
02	500000	5.70E-001	Yes
03	400000	Not converged	Yes
04	700000	Not converged	Yes
05	700000	Not converged	Yes
06	700000	8.76	Yes
07	500000	6.76E+000	Yes
08	900000	Not converged	Yes
09	500000	Not converged	No
10	300000	11.12	Yes
11	500000	3.41	No
12	900000	6.44	Yes
13	900000	12.39	Yes
14	700000	Not converged	Yes
15	700000	7.56	Yes
16	750000	2.03E+000	No
17	110000	1.84E+000	No
18	900000	5.60E+000	Yes
19	No trend	No trend	N/A
20	500000	0.66	Yes
21	300000	2.17E+000	Yes
22	700000	Not converged	No
23	300000	5.58E+000	Yes
24	150000	6.90E+000	No
25	No trend	No trend	N/A
26	700000	2.77	Yes
27	130000	0.97	No
28	700000	Not converged	Yes

The maximum deviation from the mean within a windowed range

Having established a cheaper but less representative metric which could account for variability in the data, a new metric which was a compromise between the representativeness of the standard deviation and the ease of computing of the domain

of the window was sought. The maximum deviation from the average of windowed ranges was evaluated for each sample. It was found that twenty two of the twenty eight samples could have their convergence described. It was noted that sixteen of the twenty two continued to provide the correct answer after the convergence mark had been met. This can be seen in figure 5.19 below which depicts the max deviation from the window average for windows of 500000 steps taken at 5000 step intervals for the uptake data for test set 10 depicted in figure 5.18 below. Test set 10 is observed to converge at 300000 GCMC Steps. This is described in figure 5.19 below as the metric crosses a threshold value of 17.8.

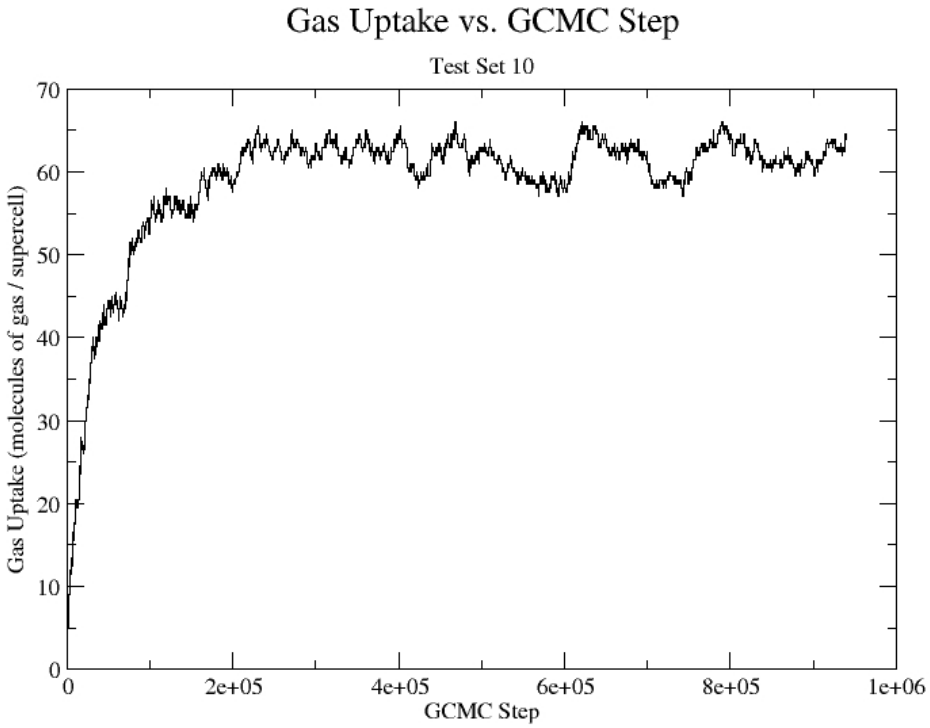


Figure 5.18 Plot of gas uptake for test set 10, showing molecules of gas adsorbed by the MOF supercell throughout the progression of the GCMC simulation steps.

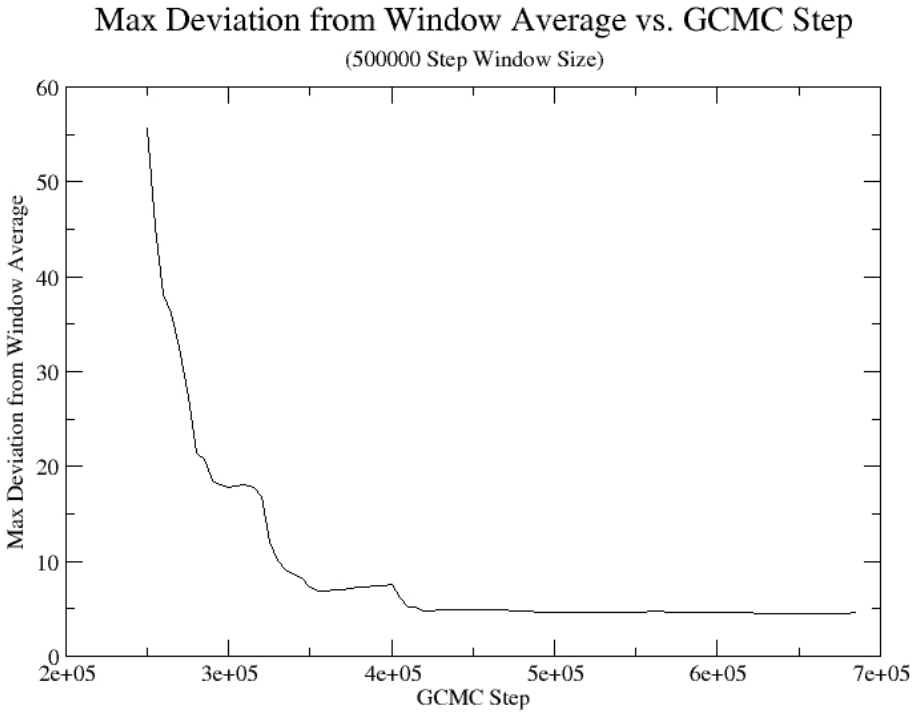


Figure 5.19 Plot of the maximum deviation from window average for the gas uptake in the simulation of test set 10 over the course of the GCMC simulation steps. The deviations are generated for windows of 500000 steps evaluated at intervals of 5000 steps.

Alternatively, samples such as test set 19, depicted in figure 5.21, showed no definitive trend by which to indicate the point of convergence for its associated uptake data, depicted in figure 5.20, had been achieved.

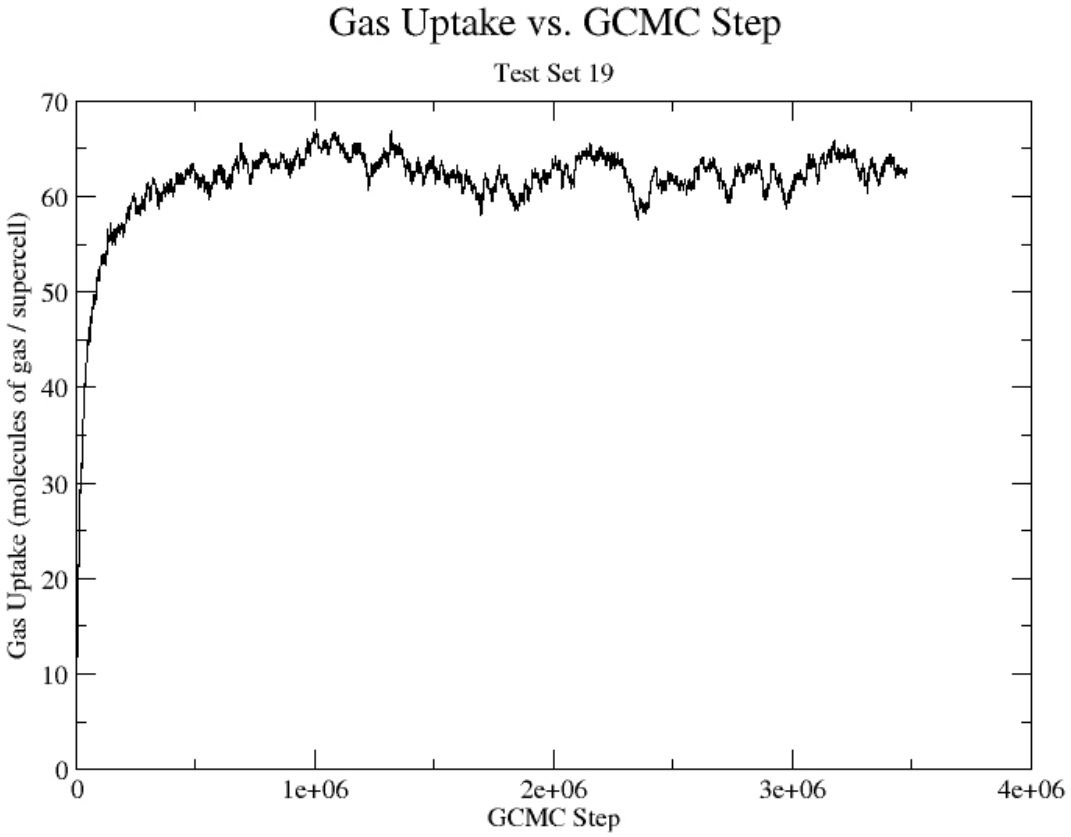


Figure 5.20 Plot of gas uptake for test set 19, showing molecules of gas adsorbed by the MOF supercell throughout the progression of the GCMC simulation steps.

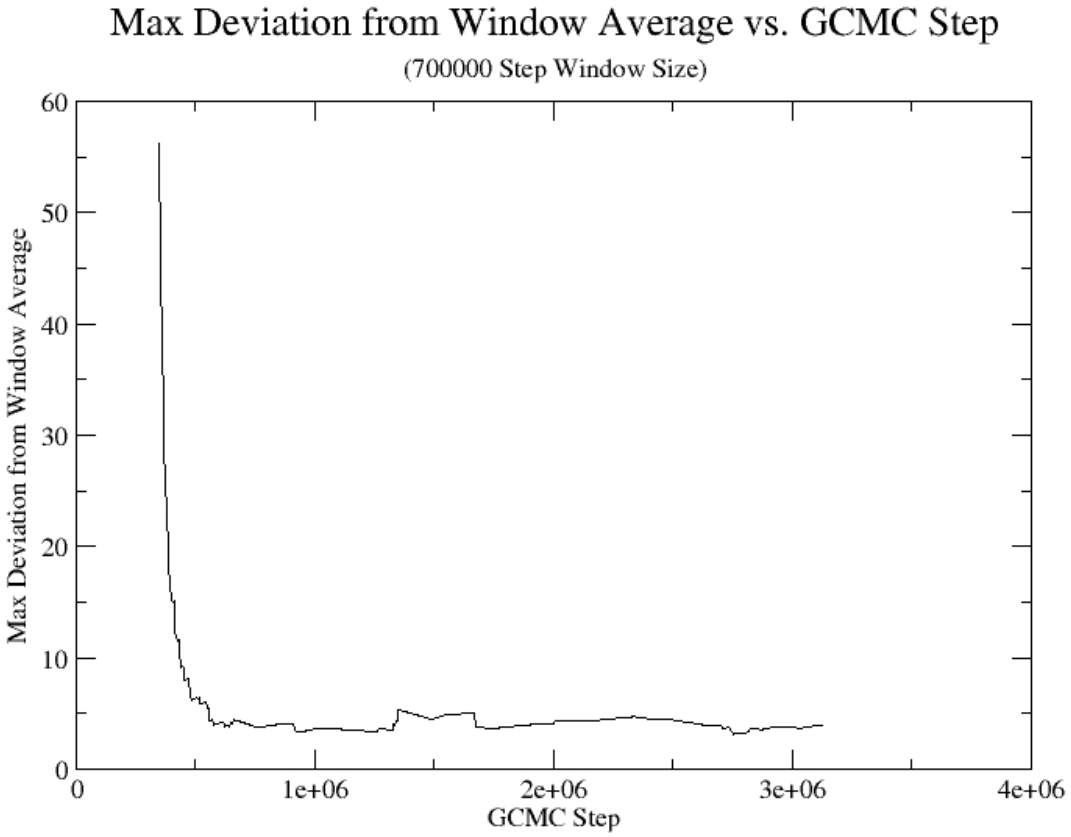


Figure 5.21 Plot of the maximum deviation from windowed average for the gas uptake in the simulation of test set 19 over the course of the GCMC simulation steps. The maximum deviations are generated for windows of 700000 steps evaluated at intervals of 5000 steps.

With regard to the negative control sets, three of the nine non-converged sets showed no trend, indicating that even should their respective simulations continue to run, it is unlikely that a the metric would serve to indicate the convergence point. As with the other metrics, no universal threshold value predicted the point of convergence across all samples. The size of the window used for the maximum deviation from the

average value varied as did the threshold. In order for the metric to be useful, it needs to be universally applicable to all simulations. The parameter values used to describe convergence using the windowed domain method is shown in table 5.5 below.

Table 5.5 Characterization of convergence assessment parameters for max deviation from window average convergence metric tested on test sets 01-28.

Sample ID	Best window size	Metric threshold value	Trend continues?
01	No trend	No Trend	N/A
02	750000	1.89E+000	Yes
03	350000	Not converged	No
04	300000	Not converged	Yes
05	700000	Not converged	Yes
06	700000	4.28	No
07	No trend	No Trend	N/A
08	700000	Not converged	No
09	700000	Not converged	Yes
10	500000	1.78E+001	Yes
11	330000	1.68	No
12	900000	3.52	Yes
13	700000	5.29	No
14	700000	Not converged	Yes
15	700000	6.16	Yes
16	930000	1.19E+000	Yes
17	590000	1.74E+000	Yes
18	No trend	No Trend	N/A
19	No trend	No Trend	N/A
20	700000	1.71	Yes
21	300000	1.11E+000	Yes
22	500000	Not converged	Yes
23	300000	3.42E+000	Yes
24	No trend	No Trend	N/A
25	No trend	No Trend	N/A
26	700000	1.76	Yes
27	130000	5.20E-001	Yes
28	900000	Not converged	No

In an attempt to achieve a relative standard, it was realized that a metric which is not heavily influenced by the magnitude of the uptake was needed to characterized the system.

The frequency of new maximum values along successive points:

The last metric explored differed in nature completely from those previously discussed. A running maximum value was taken for the simulation. As it progressed, the maxima were recorded and remapped into a space showing the distance between each successive maximum. Figure 5.22 below depicts such a scenario for test set 2, where the point of convergence coincides with a distance between successive maxima of >400000 GCMC steps.

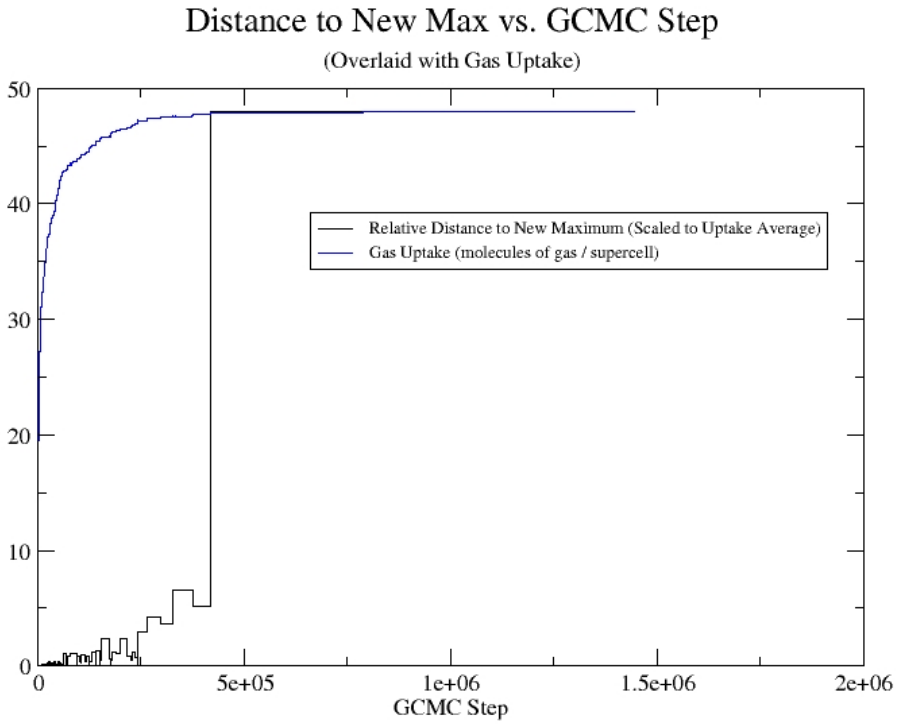


Figure 5.22 Plot of gas uptake for test set 2 showing molecules of gas adsorbed by the MOF supercell overlaid with the average uptake adjusted distance to new maximum throughout the progression of the GCMC simulation steps.

The plot continues to describe the convergence beyond this point.

Below, depicted in figure 5.23 is depicted the uptake of test set 3 overlaid with the distance between successive maxima, however, the maxima distance set as the threshold was inadequate to indicate that the system had converged. Moreover, the trend of the convergence does not seem to continue, instead decreasing in distance between distances beyond the expected point.

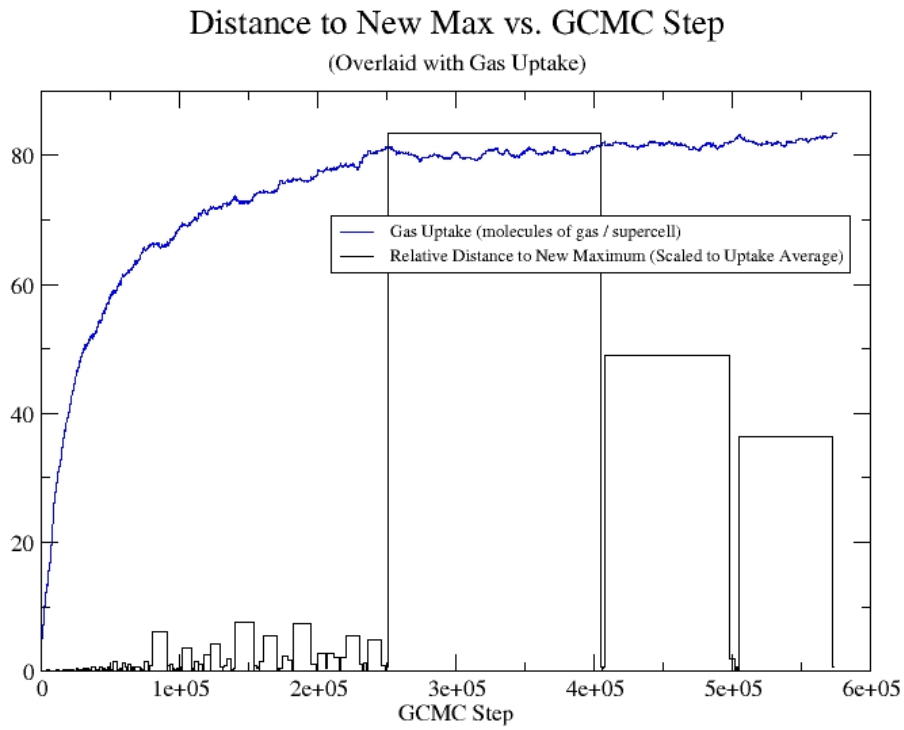


Figure 5.23 Plot of gas uptake for test set 3 showing molecules of gas adsorbed by the MOF supercell overlaid with the average uptake adjusted distance to new maximum throughout the progression of the GCMC simulation steps.

This metric was found to characterize the system with the greatest success overall. In the initial phase of the simulation, new maximum values are acquired quite quickly, however, as the simulation progresses, the acquisition of a new maximum value does not occur as frequently. For each simulation, the largest gap between maxima that did not convey convergences was recorded. It was found that this information could be used to implement a strategy for describing convergence across each set that was universally applicable and predicted the convergence and non-

convergence quite accurately, both in the converged and non-converged sets. The position of new maximum uptake values are recorded until greater than 400000 GCMC steps pass without a new maximum value being produced. Using this method, it was seen that of the nineteen converged sample sets thirteen sets predicted the convergence point within an appreciable amount of steps. Only one sample was unable to be characterized as converged given the criteria. Seven sets did not possess the extent of information needed to suggest convergence or non-convergence, and all of the eight negative control non-converge sets did not show convergence. It was seen that in every case, the convergence point could be described, within approximately 50,000 steps on average, a number of steps evaluable in a reasonable amount of time (conservatively within 20 minutes in the previously mentioned example 276 atom ZIF-8 system) using *fastmc*.

The parameter values used to describe convergence using the new-maximum frequency method is shown in table 5.5 below.

Table 5.5 Characterization of convergence assessment parameters for new-maximum frequency convergence metric tested on test sets 01-28.

Sample ID	Required Distance Between Maxima	Predicted Step
01	311000	1000000
02	50000	418000
03	154000	Not converged
04	89000	Not converged
05	157000	Not converged
06	302000	700000
07	129000	4.60E+005
08	303000	Not converged
09	368000	Not converged
10	42000	5.50E+005
11	18000	50000
12	301000	Inconclusive
13	292000	Inconclusive
14	113000	Not converged
15	36000	625000
16	225000	2.97E+006
17	353000	9.49E+005
18	189000	Inconclusive
19	279000	Inconclusive
20	77000	Inconclusive
21	115000	Inconclusive
22	438000	Not converged
23	31000	2.23E+005
24	17000	5.00E+005
25	680000	2.10E+006
26	149000	Inconclusive
27	35000	3.50E+004
28	120000	Not converged

5.5 – Conclusions

Table 5.6 – Quantitative overview of metric analysis results.

Metric (n = 28)	Sets Not Described by Metric	Value / Window Size Consistent?	Negative Controls with Trend (n = 8)	Continue Trend Beyond POC
1	7	No	7	14
2	5	No	7	15
3	3	No	6	18
4	6	No	5	16
5	1	Yes	9	N/A

Table 5.6 above shows descriptive statistics for each metric. The number of sets not described by the metric column indicates the number of sets with which the metric was unable to provide a combination of threshold value and window size which could accurately describe the accepted convergence point in the raw data. For those sets which were able to be described by some combination of value threshold and window size, the "Value / Window Size Consistent?" column indicates whether a nearly consistent value was observed in most cases. For all metrics however it was observed that both the window size and value threshold needed to be changed in all cases without ascertainable trend to accurately describe the accepted point of convergence in the uptake data. In addition the number of negative controls whereby the metric did not infer that the simulation had come to a point of convergence is depicted in the "Negative Controls with Trend" column of table 5.1. Lastly, for those sets that did reach convergence, the "Continue Trend Beyond POC" column indicates the nature of the metric beyond the point of conversion. That is to say that in some

cases, the metric continued to remain below the threshold value beyond the point of conversion while others did not.

Overall, it was found that the frequency of progression of the maximum uptake value provided an excellent metric which was able to characterize the convergence of the uptake in all simulations in a set of sample simulations selected to represent a variety of characteristic GCMC simulation results. Overall, a consistent trend among windowed metric evaluations was not attainable. This was indicated by the variation of threshold values and window sizes needed to describe the accepted point of convergence in each case. Although, it is ascertainable that the size of the system and supercell likely play a role in the determination of a threshold value and appropriate window size, such an analysis was beyond the scope of this project, and practically unfeasible due to the exceptionally large amount of data that would need to be generated. The frequency of new maximum acquisition metric (5) is favoured over the rest of those described in this section as it does not seem to depend on the size and nature of the simulation in question and provides a reasonably accurate point of convergence for the GCMC simulations used. Although they were not represented in the sample output sets, it is speculated that there may be specific cases where this metric is not applicable. In the event that the gas uptake initially exceeds the equilibrium value and decreases towards a steady state, it is expected that the frequency of maximum convergence metric would not be able to accurately describe the point of convergence. To this affect, we acknowledge that not all scenarios and possible metrics were covered within the scope of this work and future work may be best directed towards extending the evaluation of convergence by the metrics

presented. Specifically, tailoring the frequency of maximum convergence metric to an analogous scheme which depends on the minimum value may provide insight into overcoming the previously mentioned expected limitations of the maximum convergence metric. Overall, the metrics presented provide a clear foot hold in better characterizing the point of convergence within GCMC simulations, although further work is needed to clarify the relationship between the different components of the analysis in order to truly discount the utility of any of the metrics presented.

Chapter 6 - Final Remarks, Future Directions

The DOAP software presents a framework for working with simulation packages which allows the user to perform tasks in a much more straightforward manner. Applied to the task of determining the gas uptake of a metal organic framework, it provides a means of dramatically speeding up the acquisition of data and consequently the exploration of new framework structures. The scope of this project is to provide a framework by which the task of gas acquisition in metal organic frameworks can be facilitated for the user and as such it does not guarantee the certainty of the results. Methodological considerations and consequently validity of the overall simulation model are the responsibility of the user. The software simply provides a means by which to more easily execute such a simulation. That being said, it is hoped that the tools provided in the software will enable others to accomplish complex tasks with greater ease and success and that the nature of the code serves as an example in any possible extension of an equally clean, concise and easily interpretable implementations of the same nature. As it stands, the DOAP software is feature poor, but is open to change. Future directions associated with the development of this code, may look to implement features such as a job scheduler for constructing entire isotherms, support for VASP, SIESTA and other ESP packages, support for multiple guest files, the marrying of the DOAP system with a *de-novo* framework generator (spurring from influences of similar work[56,57]), improved convergence assessment metrics, a symmetry parsing utility for denoting frameworks segment parts and assigning equal atoms, support for CIF files and the like.

The complete automation of the gaseous uptake of metal organic frameworks required that a suitable convergence criteria be found in order to completely avoid an user interaction. The discovery of an appropriate metric is a large hurdle to overcome in being able to assess the completed state of the simulation. The maximum frequency evaluation method discussed in section 5.4 provides an excellent candidate for further testing in order to implement a completely automated system.

Appendix A - Program Code

chem_constants.py

```
ELEMENTS =
["H","He","Li","Be","B","C","N","O","F","Ne","Na","Mg","Al","Si","P","S","Cl","Ar","K","Ca","Sc"
,"Ti","V","Cr","Mn","Fe","Co","Ni","Cu","Zn","Ga","Ge","As","Se","Br","Kr","Rb","Sr","Y","Zr","N
b","Mo","Tc","Ru","Rh","Pd","Ag","Cd","In","Sn","Sb","Te","I","Xe","Cs","Ba","La","Ce","Pr","Nd"
,"Pm","Sm","Eu","Gd","Tb","Dy","Ho","Er","Tm","Yb","Lu","Hf","Ta","W","Re","Os","Ir","Pt","Au","
Hg","Tl","Pb","Bi","Th","Pa","U","Po","At","Rn","Fr","Ra","Ac","Np","Pu","Am","Cm","Bk","Cf","Es
","Fm","Md","No","Lr", "Rf","Db","Sg","Bh","Hs","Mt"]

MOLAR_MASSES ={"H":1.00794, "He":4.002602, "Li":6.941, "Be":9.012182, "B":10.811, "C":12.0107,
"N":14.0067, "O":15.9994, "F":18.9984032, "Ne":20.1797, "Na":22.98976928, "Mg":24.3050,
"Al":26.9815386, "Si":28.0855, "P":30.973762, "S":32.065, "Cl":35.453, "Ar":39.948,
"K":39.0983, "Ca":40.078, "Sc":44.955912, "Ti":47.867, "V":50.9415, "Cr":51.9961,
"Mn":54.938045, "Fe":55.845, "Co":58.933195, "Ni":58.6934, "Cu":63.546, "Zn":65.38, "Ga":69.723,
"Ge":72.64, "As":74.92160, "Se":78.96, "Br":79.904, "Kr":83.798, "Rb":85.4678,
"Sr":87.62, "Y":88.90585, "Zr":91.224, "Nb":92.90638, "Mo":95.96, "Tc":98, "Ru":101.07,
"Rh":102.90550, "Pd":106.42, "Ag":107.8682, "Cd":112.411, "In":114.818, "Sn":118.710,
"Sb":121.760, "Te":127.60, "I":126.90447, "Xe":131.293, "Cs":132.9054519, "Ba":137.327,
"La":138.90547, "Ce":140.116, "Pr":140.90765, "Nd":144.242, "Pm":145, "Sm":150.36, "Eu":151.964,
"Gd":157.25, "Tb":158.92535, "Dy":162.500, "Ho":164.93032, "Er":167.259, "Tm":168.93421,
"Yb":173.054, "Lu":174.9668, "Hf":178.49, "Ta":180.94788, "W":183.84,
"Re":186.207, "Os":190.23, "Ir":192.217, "Pt":195.084, "Au":196.966569, "Hg":200.59,
"Tl":204.3833, "Pb":207.2, "Bi":208.98040, "Po":209, "At":210, "Rn":222, "Fr":223, "Ra":226,
"Ac":227, "Th":232.03806, "Pa":231.03588, "U":238.02891, "Np":237, "Pu":244,
"Am":243, "Cm":247, "Bk":247, "Cf":251, "Es":252, "Fm":257, "Md":258, "No":259, "Lr":262,
"Rf":265, "Db":268, "Sg":271, "Bh":272, "Hs":270, "Mt":276, "Ds":281, "Rg":280, "Cn":285,
"Uut":284, "Uuq":289, "Uup":288, "Uuh":293, "Uuo":294}

ATOMIC_NUMBER =
{1:'H',2:'He',3:'Li',4:'Be',5:'B',6:'C',7:'N',8:'O',9:'F',10:'Ne',11:'Na',12:'Mg',13:'Al',14:'Si
',15:'P',16:'S',17:'Cl',18:'Ar',19:'K',20:'Ca',21:'Sc',22:'Ti',23:'V',24:'Cr',25:'Mn',26:'Fe',27
:'Co',28:'Ni',29:'Cu',30:'Zn',31:'Ga',32:'Ge',33:'As',34:'Se',35:'Br',36:'Kr',37:'Rb',38:'Sr',39
:'Y',40:'Zr',41:'Nb',42:'Mo',43:'Tc',44:'Ru',45:'Rh',46:'Pd',47:'Ag',48:'Cd',49:'In',50:'Sn',51:
'Sb',52:'Te',53:'I',54:'Xe',55:'Cs',56:'Ba',57:'La',58:'Ce',59:'Pr',60:'Nd',61:'Pm',62:'Sm',63:'
Eu',64:'Gd',65:'Tb',66:'Dy',67:'Ho',68:'Er',69:'Tm',70:'Yb',71:'Lu',72:'Hf',73:'Ta',74:'W',75:'R
e',76:'Os',77:'Ir',78:'Pt',79:'Au',80:'Hg',81:'Tl',82:'Pb',83:'Bi',84:'Th',85:'Pa',86:'U',87:'Po
',88:'At',89:'Rn',90:'Fr',91:'Ra',92:'Ac',93:'Np',94:'Pu',95:'Am',96:'Cm',97:'Bk',98:'Cf',99:'Es
',100:'Fm',101:'Md',102:'No',103:'Lr',104:'Rf',105:'Db', 106:'Sg',107:'Bh',108:'Hs'}

MIN_SIGMA = 2.0
MAX_SIGMA = 4.5
MIN_EPSILON = 0.004
MAX_EPSILON = 0.800
```

cpmd_tools.py

```
import os
import doap_logging
import xyz_tools
import commands

log = doap_logging.getLogger(__name__)


def string_to_bool(string):
    if string.upper() == "TRUE":
        return True
    elif string.upper() == "FALSE":
        return False
    else:
```

```
        raise ValueError


def is_binary_data(data):
    return ( sum([1 for i in data if i < "\x20" or i > "\x7A"]) /float( len(data) ) ) > 0.80

def read_file_header(filename):
    tempfile = open(filename, 'r')
    dump = tempfile.read(2048)
    tempfile.close()
    return dump

def is_binary_psp_file(filename):
    return is_binary_data(read_file_header(filename))

def count_all_non_hydrogen_atoms(atom_counts,atom_labels):
    return sum([atom_counts[atom] for atom in atom_labels if atom != "H"])

def get_psp_files(cpmd_psp_path):
    log.info("Looking for Pseudo Potential files in: " + cpmd_psp_path)
    return [file for file in os.listdir(cpmd_psp_path) if os.path.isfile(cpmd_psp_path + file)]

def select_new_psp_file(atom_symbol, atom_labels, psp_files, cpmd_psp_type, cpmd_functional):
            log.info("\nDefault \"" + atom_labels[i] + "_" + cpmd_psp_type + "_" +
cpmd_functional + ".psp\" not found, please select from the list below:")
            sublist = [file for file in psp_files if atom_symbol in file]
            for j in range(len(sublist)):
                print ("Entry %(acc_j)3i: " % {'acc_j': j} + sublist[j])

            choice = raw_input("Enter index number:")
            choice_valid = False
            while not(choice_valid):
                try:
                    if (int(choice))in range(len(sublist)):
                        choice_valid = True
                    else:
                        print("Invalid index choice, please choose again\n")
                        choice = raw_input("Enter index number:")
                except:
                    print("Invalid index choice, please choose again\n")
                    choice = raw_input("Enter index number:")
            return sublist[int(choice)]

def confirm_psp_choices(atom_labels, psp_files, cpmd_psp_path, cpmd_psp_type, cpmd_functional):
    psp_choices = []
    for i in range(len(atom_labels)):
        if (os.path.exists(cpmd_psp_path + atom_labels[i] + "_" + cpmd_psp_type +
"_"+cpmd_functional + ".psp")):
            psp_choices.append(atom_labels[i] + "_" + cpmd_psp_type + "_" + cpmd_functional +
".psp")
        elif (os.path.exists(cpmd_psp_path + atom_labels[i] + "_" + cpmd_psp_type + "_" +
cpmd_functional)):
            psp_choices.append(atom_labels[i] + "_" + cpmd_psp_type + "_" + cpmd_functional)
        else:
            psp_choice.append(select_new_psp_file(atom_symbol, atom_labels, psp_files))
    if len(psp_choices) == 0:
        log.error("Failed to generate list of psp choices. Error.")
    return psp_choices

def vdb_present(psp_choices):
    return sum([1 for psp_choice in psp_choices if "VAN" in psp_choice or "VDB" in psp_choice])
> 0

def assign_binary_or_formated_to_psps(psp_choices, cpmd_psp_path):
    temp_psp_choices = list(psp_choices)
    binary_psp_mask = [is_binary_psp_file(cpmd_psp_path + '/' + filename) for filename in
temp_psp_choices]
    return [filename + ' ' + (isbinary and 'BINARY' or 'FORMATTED') for (filename, isbinary) in
zip(temp_psp_choices, binary_psp_mask)]
```

```
def hydrogens_present(atom_labels):
    return [1] == [1 for x in atom_labels if x.upper() == "H"]

def write_cpmd_section(atom_labels, hydrogens_present, cpmd_input_file,
cpmd_convergence_crit_orbitals, cpmd_max_iterations, cpmd_optimize_hydrogens):
    cpmd_section = "&CPMD\n  OPTIMIZE WAVEFUNCTION XYZ\n  CONVERGENCE ORBITALS\n  "\
                   + str(cpmd_convergence_crit_orbitals) + "\n  CONVERGENCE GEOMETRY\n  5.0d-4\n
MAXITER\n  "\
                   + str(cpmd_max_iterations) + "\n  PCG MINIMIZE\n  ELECTROSTATIC
POTENTIAL\n&END\n"

    if cpmd_optimize_hydrogens and hydrogens_present:
        cpmd_section = cpmd_section.replace("OPTIMIZE WAVEFUNCTION XYZ", "OPTIMIZE GEOMETRY
WAVEFUNCTION XYZ")
    cpmd_input_file.write(str(cpmd_section))

def write_dft_section(cpmd_input_file, cpmd_functional):
    dft_section = "\n&DFT\n    FUNCTIONAL " + cpmd_functional + "\n&END\n"
    cpmd_input_file.write(dft_section)


def write_system_section(cpmd_input_file, unit_cell_vectors, psp_choices, cpmd_cutoff,
cpmd_charge, cpmd_check_for_dual, cpmd_vdb_dual):
    system_section1 = "&SYSTEM\n  ANGSTROM\n CELL VECTOR\n"
    system_section2 = "  CUTOFF\n"+str(cpmd_cutoff)+"\n  CHARGE\n"+cpmd_charge+"\n&END\n\n"
    cpmd_input_file.write(str(system_section1))
    def write_cpmd_vector_line(unit_cell_vector_coordinate_list):
        x, y, z = unit_cell_vector_coordinate_list
        cpmd_input_file.write("%20.12f  %20.12f  %20.12f\n" % (x, y, z))
    [write_cpmd_vector_line([x, y, z]) for [CV, x, y, z] in unit_cell_vectors]
    if vdb_present(psp_choices) and cpmd_check_for_dual:
        cpmd_input_file.write("DUAL " + str(cpmd_vdb_dual) + "\n")
    cpmd_input_file.write(str(system_section2))


def write_atoms_section(cpmd_input_file, atom_labels, psp_choices, atom_counts,
ordered_coordinates,_hydrogens_present, cpmd_optimize_hydrogens, cpmd_lmax_flag):

    cpmd_input_file.write("&ATOMS\n")
    def write_cpmd_atom_line(atom_line):
        [x, y, z] = atom_line
        cpmd_input_file.write(str(x).rjust(20) + str(y).rjust(20) + str(z).rjust(20) + "\n")

    for i in range(len(atom_labels)):
        cpmd_input_file.write("*" + str(psp_choices[i]) + "\nLMAX=" + str(cpmd_lmax_flag) + "\n"
+ str(atom_counts[atom_labels[i]]) + "\n")
        [write_cpmd_atom_line([x, y, z]) for [atom_symbol, x, y, z] in ordered_coordinates if
atom_symbol == atom_labels[i]]

    cpmd_input_file.write("&END\n")

    if cpmd_optimize_hydrogens and hydrogens_present:
       cpmd_input_file.write("&CONSTRAINTS\n   FIX SEQUENCE\n    1 " +
str(count_all_non_hydrogen_atoms(atom_counts, atom_labels)) + "\n&END CONSTRAINTS")
    elif not hydrogens_present(atom_labels):
        log.info("No Hydrogens found to optimize in system.")
    elif not cpmd_optimize_hydrogens:
        log.info("Hydrogen optimization not selected.")

def write_cpmd_input_file(atom_labels, atom_counts, cpmd_input_file, ordered_coordinates,
unit_cell_vectors, psp_choices, cpmd_convergence_crit_orbitals, cpmd_max_iterations,
cpmd_optimize_hydrogens, cpmd_functional, cpmd_cutoff, cpmd_charge, cpmd_lmax_flag,
cpmd_check_for_dual, cpmd_vdb_dual):
    _hydrogen_present =  hydrogens_present(atom_labels)
    write_cpmd_section(atom_labels, _hydrogen_present, cpmd_input_file,
cpmd_convergence_crit_orbitals, cpmd_max_iterations, cpmd_optimize_hydrogens)
    write_dft_section(cpmd_input_file, cpmd_functional)
    write_system_section(cpmd_input_file, unit_cell_vectors, psp_choices, cpmd_cutoff,
cpmd_charge, cpmd_check_for_dual, cpmd_vdb_dual)
```

```
    write_atoms_section(cpmd_input_file, atom_labels, psp_choices, atom_counts,
ordered_coordinates, _hydrogen_present, cpmd_optimize_hydrogens, cpmd_lmax_flag)


def import_unit_vectors_from_cpmd_input_file(cpmd_input_file_name):
    unit_cell_vector_lines = commands.getoutput('grep VECTOR -A 3 ' + cpmd_input_file_name + ' |
grep -v VECTOR')
    split_unit_cell_vector_lines = [line.split() for line in unit_cell_vector_lines.split("\n")]
    unit_cell_vectors = [ [float(x), float(y), float(z)] for [x, y, z] in
split_unit_cell_vector_lines]
    return unit_cell_vectors

def get_four_columned_lines_from_cpmd_geometry_file(line_list):
    return [[x for x in y.split()[:4]] for y in line_list if len(y.split()) == 7] #Geometry
files have 7 items, atom, x, y, z and forces in x, y, z.

def extract_cpmd_geometry_atom_lines(line_list):
    return [[symbol, float(x), float(y), float(z)] for [symbol, x, y, z] in line_list]

def parse_cpmd_geometry_file(filename):
    try:
        input_file=open(filename, 'r')
        line_list = input_file.readlines()
        input_file.close()
    except IOError,e:
        raise e
    four_columned_lines = get_four_columned_lines_from_cpmd_geometry_file(line_list)
    atom_coordinates = extract_cpmd_geometry_atom_lines(four_columned_lines)
    return atom_coordinates

def create_cpmd_input_file(xyz_input_file_name, cpmd_job_name, charge_config):

    cpmd_check_for_dual = True
    cpmd_vdb_dual = 6
    cpmd_convergence_crit_orbitals = charge_config["cpmd_convergence_crit_orbitals"]
    cpmd_max_iterations = charge_config["cpmd_max_iterations"]
    cpmd_cutoff = charge_config["cpmd_cutoff"]
    cpmd_psp_path = charge_config["cpmd_psp_path"]
    cpmd_functional = charge_config["cpmd_functional"]
    cpmd_psp_type  = charge_config["cpmd_psp_type"]
    cpmd_lmax_flag  = charge_config["cpmd_lmax_flag"]
    cpmd_charge = charge_config["general_system_charge"]
    cpmd_optimize_hydrogens = string_to_bool(charge_config["cpmd_optimize_hydrogens"])

    cpmd_input_file=open(cpmd_job_name+".in", 'w')

    (atom_coordinates, unit_cell_vectors) = xyz_tools.parse_unit_cell_file(xyz_input_file_name)

    atom_coordinates = xyz_tools.reassign_proper_atom_labels(atom_coordinates)

    atom_labels = xyz_tools.create_atom_label_set(atom_coordinates)
    atom_labels = xyz_tools.reorder_H_atoms_to_last(atom_labels)
    atom_counts = xyz_tools.get_atom_quantities(atom_coordinates, atom_labels)

    log.info("\nSystem described as:")
    log.info(str(atom_labels))
    log.info(str(atom_counts))

    psp_files = get_psp_files(cpmd_psp_path)
    psp_choices = confirm_psp_choices(atom_labels, psp_files, cpmd_psp_path, cpmd_psp_type,
cpmd_functional)

    log.info("\nThe following PseudoPotentials have been selected.")
    log.info(str(psp_choices))

    formatted_psp_choices = assign_binary_or_formated_to_psps(psp_choices, cpmd_psp_path)

    ordered_coordinates, index = xyz_tools.index_and_sort_coordinate_list(atom_coordinates,
atom_labels)
    log.info("Writing CPMD input file . . .")
```

```
    write_cpmd_input_file(atom_labels, atom_counts, cpmd_input_file, ordered_coordinates,
unit_cell_vectors, psp_choices, cpmd_convergence_crit_orbitals, cpmd_max_iterations,
cpmd_optimize_hydrogens, cpmd_functional, cpmd_cutoff, cpmd_charge, cpmd_lmax_flag,
cpmd_check_for_dual, cpmd_vdb_dual)

    log.info("\nWriting ORDER file")
    order_file_name = cpmd_job_name+".atom_order"
    xyz_tools.write_atom_order_to_file(index,order_file_name)

    cpmd_input_file.close()
```

data_validation.py

```
import re
import os
import doap_logging
import fastmc_tools
import xyz_tools
import doap_tools
from validation_criteria import CRITERIA
from validation_criteria import TYPE
from validation_criteria import REGEX
from chem_constants import MIN_SIGMA
from chem_constants import MAX_SIGMA
from chem_constants import MIN_EPSILON
from chem_constants import MAX_EPSILON
from exception_classes import *



log = doap_logging.getLogger(__name__)

def all(iterable):
    for element in iterable:
        if not element:
            return False
    return True

def validate_job_config_list_shape(job_config_list):
    try:
        [[key, value] for [key, value] in job_config_list]
    except:
       job_config_has_incorrect_shape

def value_agrees_with_regular_expression(value, expression):
    return re.match(expression, value) != None

def regex_for_key(key):
    return CRITERIA[key.upper()][REGEX]

def type_for_key(key):
    return CRITERIA[key.upper()][TYPE]


def value_can_be_casted_to_correct_type(value, type):
    try:
        type(value)
        return True
    except ValueError:
        return False

def validate_all_keys_for_type(dictionary):
    return all([value_can_be_casted_to_correct_type(dictionary[key], type_for_key(key)) for key
in dictionary.keys()])

def validate_all_keys_for_regex(dictionary):
    return all([value_agrees_with_regular_expression(dictionary[key], regex_for_key(key)) for
key in dictionary.keys()])
```

```
def validate_config_arguments(main_config_dictionary):
    if not validate_all_keys_for_type(main_config_dictionary) and
validate_all_keys_for_regex(main_config_dictionary):
       raise ConfigNotValidException


def validate_guest_file(guest_file_name):
    guest_coordinates, guest_field_section, guest_vdw_data =
fastmc_tools.import_fastmc_guest_file_components(guest_file_name)
    coordinate_atom_symbols =  xyz_tools.create_atom_label_set(guest_coordinates)
    field_atom_symbols =  xyz_tools.create_atom_label_set(guest_field_section)
    vdw_atom_symbols =  xyz_tools.create_atom_label_set(guest_vdw_data)

    if len(guest_coordinates) != len(guest_field_section):
        log.error("Guest file formatted incorrectly: field parameters incomplete")
        raise GuestFieldParametersIncompleteException

    for atom_symbol in coordinate_atom_symbols:
        if not (atom_symbol in field_atom_symbols):
            log.error("Atom: " + atom_symbol + " is missing some field info.")
            raise AtomMissingFieldInfoException

        if not (atom_symbol in vdw_atom_symbols):
            log.error("Atom: " + atom_symbol + " is missing some vdw info.")
            raise AtomMissingVdwInfoException


def validate_vanderWaals_file(vdw_file_name):

    vdw_file=open(vdw_file_name,'r')
    vdw_lines = vdw_file.readlines()
    vdw_file.close()

    for vdw_line in vdw_lines:

        split_vdw_line = vdw_line.split()

        if len(split_vdw_line) != 3:
            log.error("vdw file is formatted incorrectly: line(s) with >/< 3 elements")
            raise BadVdwLineException

        if not all( map( xyz_tools.is_floatable, split_vdw_line[1:] ) ):
            log.error("vdw file is incorrectly formatted: line " + split_vdw_line[0] + " has non
numerical values.")
            raise BadVdwLineException

    vdw_values = [[line.split()[0],map(float,line.split()[1:])] for line in vdw_lines]
    epsilons = [epsilon for [label, [sigma, epsilon]] in vdw_values]
    sigmas = [sigma for [label, [sigma, epsilon]] in vdw_values]
    if not ( min(sigmas) > MIN_SIGMA and max(sigmas) < MAX_SIGMA and min(epsilons) > MIN_EPSILON
and max(epsilons) < MAX_EPSILON ):
        log.warning("sigma and epsilon values are outside expected ranges")
        raise VdwValuesOutOfRangeException
    log.info("vanderWaals file validated")


def validate_xyz_input_file(xyz_input_file_name):
    input_file=open(xyz_input_file_name, 'r')
    line_list = input_file.readlines()
    input_file.close()
    four_columned_lines = xyz_tools.get_four_columned_lines(line_list)
    line_list = [line for line in line_list if line != "\n"]
    atom_coordinates = xyz_tools.extract_lines_by_symbol_filter(xyz_tools.atom_symbol_filter,
four_columned_lines)
    unit_cell_vectors = xyz_tools.extract_lines_by_symbol_filter(xyz_tools.cell_vector_filter,
four_columned_lines)

    if len(atom_coordinates)+len(unit_cell_vectors) != len(line_list):
        log.warning("Lines present in input file not formatted for unit cells or coordinates")
```

```
    if len(unit_cell_vectors) != 3:
        log.error("Input file has incorrect number of unit cell vectors lines")
        raise IncorrectUnitCellLinesException

    atom_label_list = xyz_tools.create_atom_label_set(atom_coordinates)
    proper_atom_labels = xyz_tools.find_proper_atom_labels(atom_label_list)
    if None in proper_atom_labels:
        log.error("Unable to resolve all atom labels.")
        raise NotAllAtomLabelsResolvableException
    log.info("xyz file passed validation")


def validate_paths_exist(list_of_paths):
    if all([os.path.exists(path) for path in list_of_paths]):
        pass
    else:
        raise PathMissingException

def validate_files_exist(list_of_files):
    if  all([os.path.exists(file) for file in list_of_files]):
        pass
    else:
        raise FileMissingException



def return_smallest_cell_dimension(unit_cell_vectors):
    temp_unit_cell_vector_list = list(unit_cell_vectors)
    vectors_only_list = [[x,y,z] for [symbol, x, y, z] in temp_unit_cell_vector_list]
    squared_unit_cell_vector_list = [[n**2 for n in ucv_line] for ucv_line in vectors_only_list]
    unit_cell_vector_lengths = [ sum(squared_ucv_line)**0.5 for squared_ucv_line in
squared_unit_cell_vector_list]
    return  min(unit_cell_vector_lengths)


def validate_files_and_paths_exist(list_of_files, list_of_paths):
    validate_files_exist(list_of_files)
    validate_paths_exist(list_of_paths)


def validate_cell_cutoff(config_dictionary, xyz_input_file_name):
    cutoff = config_dictionary["gcmc_cutoff"]
    (atom_coordinates, unit_cell_vectors) = xyz_tools.parse_unit_cell_file(xyz_input_file_name)
    smallest_half_cell_length = return_smallest_cell_dimension(unit_cell_vectors)/2
    print type(cutoff), type(smallest_half_cell_length)
    if float(cutoff) > smallest_half_cell_length:
        log.warning("Cutoff exceeds half cell width")
        raise CellCutoffExceedsHalfCellWidthException
    log.info("Cell cutoff validated")

def validate_config_options(config_dictionary, xyz_input_file_name):
    vdw_file_name = config_dictionary["general_vdw_file"]
    guest_file_name = config_dictionary["gcmc_guest_file"]
    list_of_files = doap_tools.return_subdictionary_containing_keys_with("FILE",
config_dictionary).values()
    list_of_paths = doap_tools.return_subdictionary_containing_keys_with("PATH",
config_dictionary).values()
    validate_files_and_paths_exist(list_of_files, list_of_paths)
    validate_cell_cutoff(config_dictionary, xyz_input_file_name)
    validate_xyz_input_file(xyz_input_file_name)
    validate_vanderWaals_file(vdw_file_name)
    validate_guest_file(guest_file_name)
```

doap_logging.py

```
import os
import logging
```

```
username = os.getlogin()

formatter = logging.Formatter(username + ':%(process)d | %(levelname)8s | %(module)14s:%
(lineno)3d @ %(relativeCreated)7d | %(asctime)s : %(message)s')

streamHandler = logging.StreamHandler()
streamHandler.setFormatter(formatter)

def getLogger(name):
    log = logging.getLogger(name)
    log.addHandler(streamHandler)
    log.setLevel(logging.DEBUG)
    return log
```

DOAP_OPTIONS

```
#----------------------
#        MAIN
#----------------------

#Package for charge determination | CPMD_REPEAT  (VASP)
&MAIN_CHARGE_PACKAGE CPMD

#Point charge resolutions package |  REPEAT
&MAIN_POINT_CHARGE_PACKAGE REPEAT

#GCMC Package to use | FASTMC
&MAIN_GCMC_PACKAGE FASTMC


#----------------------
#        GENERAL
#----------------------

#Filename for VDW params | e.g.: P
&GENERAL_VDW_FILE vanderWaal_parameters

#System charge
&GENERAL_SYSTEM_CHARGE 0

#Temperature in Kelvin
&GENERAL_SYSTEM_TEMP 273

#----------------------
#        CPMD
#----------------------

#Number of Processors to use in CPMD | e.g.: 8
&CPMD_NUM_PROCESSORS 32

#Orbital Convergence criteria in CPMD | e.g.: 1.0d-5
&CPMD_CONVERGENCE_CRIT_ORBITALS 1.0d-5

#Maximum number of optimization steps | e.g.: 10000
&CPMD_MAX_ITERATIONS 10000

#Absolute path for pseudopotential files | e.g.: /home/program/CHEMISTRY/CPMD/Pseudo_potentials/
&CPMD_PSP_PATH /home/program/CHEMISTRY/CPMD/Pseudo_potentials/

#Default functional type | e.g.: PBE
&CPMD_FUNCTIONAL PBE

#Default pseudopotential type | e.g.: SG
&CPMD_PSP_TYPE SG

#Default highest orbital consideration | e.g.: P
&CPMD_LMAX_FLAG P
```

```
#Cutoff for CPMD calculation in angstroms | e.g.: 70.0
&CPMD_CUTOFF 70.0

#Flag for whether or not to optimize Hydrogens | True or False
&CPMD_OPTIMIZE_HYDROGENS False


#----------------------
#        REPEAT
#----------------------

#Fit molecular(0) or periodic(1:default) system? | e.g.: 0
&REPEAT_PERIODICITY  1

#van der Waals scaling factor (default = 1.0) | e.g.: 1.0
&REPEAT_VDW_FACTOR 1.0

#Apply RESP penalties?, no(0:default), yes(1) | e.g.: 0
&REPEAT_RESP_PENALTIES 0

#Read cutoff radius? no(0), yes(1:default) | e.g.: 1
&REPEAT_READ_CUTOFF 1

#If flag above=1 provide R_cutoff next (in Bohrs) | e.g.: 20.0
&REPEAT_R_CUTOFF 20.0

#Apply symmetry restrain? no(0:default), yes(1) | e.g.: 0
&REPEAT_APPLY_SYMMETRY_RESTRAIN 0

#Use Goddard-type restrains? no(0:default), yes(1) | e.g.: 0
&REPEAT_USE_GODDARD 0

#If flag above=1 then provide weight next | e.g.: 0.0
&REPEAT_GODDARD_WEIGHT 0.5



#----------------------
#        GCMC
#----------------------

#Multiplicity of supercell in A direction | e.g.: 2
&GCMC_SUPERCELL_DIM_A 2

#Multiplicity of supercell in B direction | e.g.: 2
&GCMC_SUPERCELL_DIM_B 2

#Multiplicity of supercell in C direction | e.g.: 2
&GCMC_SUPERCELL_DIM_C 2

#Filename for guest information | e.g.: GUEST
&GCMC_GUEST_FILE GUEST

#Required forces cutoff (Angstroms)
&GCMC_CUTOFF  4

#Verlet neighbour list shell width (Angstroms)
&GCMC_DELR  1.0

#Ewald sum for electrostatics, with automatic parameter optimisation (0 < f < 1.E . 4)
&GCMC_EWALD_PRECISION 1d-6

#Number of Molecular types | e.g.: 2
&GCMC_MOL_TYPES 2

#See DL_POLY manual | e.g.: 3
&GCMC_IMCON 3

#----------------------
#       FASTMC
```

```
#----------------------


#Number of Guests in fast GCMC
&FASTMC_NUM_GUESTS 1

#Partial pressure of Guest
&FASTMC_GUEST_PP   0.850

#fastGCMC simulation length
# Simulation length and equilibration steps that count towards average. (over all nodes)
&FASTMC_SIM_LENGTH 1000000

#fastGCMC equilibration time [First do this number of steps (for each node)]
&FASTMC_EQ_TIME 100000
```

doap_tools.py

```
import re
import os
import sys
import data_validation
import doap_logging

log = doap_logging.getLogger(__name__)


def remove_comments_from_line_list(line_list):
    return [x[:x.find("#")] for x in line_list if len(x[:x.find("#")]) > 0]

#switches list from [ [ [a1, b1, c1], [a2, b2, c2] ] , [ [a3, b3, c3], [a4, b4, c4] ] ]  -->
[ [a1, b1, c1], [a2, b2, c2], [a3, b3, c3], [a4, b4, c4] ]
def reduce_list_depth(list):
    templist = []
    [templist.extend(i) for i in list]
    return templist

def make_job_option_dictionary(options_list):
    return dict([(x.replace("&", "").lower(), y) for [x, y] in options_list])

def get_line_list_from_file(filename):
    try:
        tempfile = open(filename, 'r')
        line_list = tempfile.readlines()
        tempfile.close()
        return line_list
    except IOError, e:
        log.error("File not found.")
        raise e

def return_lines_containing_keywords(keyword_list, line_list):
    results = [[line for line in line_list if keyword in line] for keyword in keyword_list]
    return reduce_list_depth(results)

def return_subdictionary_containing_keys_with(keyword_list, dictionary):
    results = [[[key, dictionary[key]] for key in dictionary.keys() if keyword in key] for
keyword in keyword_list]
    results = reduce_list_depth(results)
    new_dictionary = dict(results)
    return new_dictionary


def strip_lines_and_split_into_lists(line_list):
    return [x.strip().split() for x in line_list]

def get_uncommented_config_line_list_from_file(filename):
    raw_job_config_line_list = get_line_list_from_file(filename)
    uncommented_job_config_line_list = remove_comments_from_line_list(raw_job_config_line_list)
```

```
    return uncommented_job_config_line_list

def get_main_option_prefixes(uncommented_job_config_line_list):
    main_config_line_list = return_lines_containing_keywords(["MAIN"],
uncommented_job_config_line_list)
    main_config_list = strip_lines_and_split_into_lists(main_config_line_list)
    data_validation.validate_job_config_list_shape(main_config_list)
    main_config_dictionary = make_job_option_dictionary(main_config_list)
    data_validation.validate_config_arguments(main_config_dictionary)
    option_prefix_list = main_config_dictionary.values()
    return option_prefix_list

def return_effective_config_dictionary(job_config_file_name):
    uncommented_config_line_list =
get_uncommented_config_line_list_from_file(job_config_file_name)
    main_option_prefix_list = get_main_option_prefixes(uncommented_config_line_list)
    main_option_prefix_list.append("GENERAL")
    main_option_prefix_list.append("GCMC")
    config_line_list = return_lines_containing_keywords(main_option_prefix_list,
uncommented_config_line_list)
    config_list = strip_lines_and_split_into_lists(config_line_list)
    data_validation.validate_job_config_list_shape(config_list)
    config_dictionary = make_job_option_dictionary(config_list)
    return config_dictionary
```

exception_classes.py

```
class VdwValuesOutOfRangeException(Exception):
    pass

class GuestFieldParametersIncompleteException(Exception):
    pass

class AtomMissingFieldInfoException(Exception):
    pass

class AtomMissingVdwInfoException(Exception):
    pass

class BadVdwLineException(Exception):
    pass

class BadVdwLineException(Exception):
    pass

class VdwValuesOutOfRangeException(Exception):
    pass

class IncorrectUnitCellLinesException(Exception):
    pass

class NotAllAtomLabelsResolvableException(Exception):
    pass

class PathMissingException(Exception):
    pass

class FileMissingException(Exception):
    pass

class CellCutoffExceedsHalfCellWidthException(Exception):
    pass

class ConfigNotValidException(Exception):
    pass
```

fastmc_tools.py

```
#! /usr/bin/env python

import os
import doap_logging
import xyz_tools
import cpmd_tools
from numpy import product
from chem_constants import MOLAR_MASSES
from molecular_mechanics_tools import SIGMA
from molecular_mechanics_tools import EPSILON

log = doap_logging.getLogger(__name__)

def is_a_field_vdw_line(line):
    split_line = line.split()
    if not len(split_line) == 4:
        return False
    return min([split_line[0] == "VDW", split_line[1][0].isalpha()]+
[xyz_tools.is_floatable(detail) for detail in split_line[2:]])

def is_a_field_charge_weight_line(line):
    split_line = line.split()
    if not len(split_line) == 3:
        return False
    return min([split_line[0][0].isalpha()] + [xyz_tools.is_floatable(detail) for detail in
split_line[1:]])


def import_fastmc_guest_file_components(guest_file):
    try:
        guest_file=open(guest_file, 'r')
        guest_lines = guest_file.readlines()
        guest_file.close()
    except IOError, e:
        raise e

    guest_coordinates = [line.split() for line in guest_lines if
xyz_tools.is_a_coordinate_line(line)]
    guest_name = guest_lines[0]
    guest_field_section = [line.split() for line in guest_lines if
is_a_field_charge_weight_line(line)]
    guest_vdw_data = [[symbol, epsilon, sigma] for [vdw, symbol, epsilon, sigma] in
[line.split() for line in guest_lines if is_a_field_vdw_line(line)]]

    return guest_coordinates, guest_field_section, guest_vdw_data


def import_fastmc_guest_file(guest_file):
    try:
        guest_file=open(guest_file, 'r')
        guest_lines = guest_file.readlines()
        guest_file.close()
    except IOError, e:
        raise e

    guest_coordinates = [line.split() for line in guest_lines if
xyz_tools.is_a_coordinate_line(line)]
    guest_name = guest_lines[0]
    guest_field_section = [line.split() for line in guest_lines if
is_a_field_charge_weight_line(line)]
    guest_vdw_data = [[symbol, epsilon, sigma] for [vdw, symbol, epsilon, sigma] in
[line.split() for line in guest_lines if is_a_field_vdw_line(line)]]
    guest_field_data = [[symbol, weight, charge, x, y, z] for [symbol, weight, charge, symbol,
x, y, z] in [guest_field_section[guest_coordinates.index(x)] + x for x in guest_coordinates]]

    return guest_field_data, guest_name, guest_vdw_data
```

```
def write_fastmc_config_file(doap_job_name, supercell_dimensions, cell_vectors,
cell_coordinates, imcon):

    def write_fastmc_style_atom_line(coordinate_line, config_file_index):
        [curr_atom, x, y, z] = coordinate_line
        config_file.write(curr_atom.ljust(8)+"%10i\n" % config_file_index)
        config_file.write("%20.12f %20.12f %20.12f\n" % (x, y, z))
        config_file_index += 1

    def write_fastmc_style_unit_cell_vector_line(unit_cell_vector_coordinate_list):
        x, y, z = unit_cell_vector_coordinate_list
        config_file.write("%20.12f %20.12f %20.12f\n" % (x, y, z))

    config_file=open('CONFIG', 'w')
    config_file.write(doap_job_name + " " + str(supercell_dimensions[0]) + "X" +
str(supercell_dimensions[1]) + "X" + str(supercell_dimensions[2]) + "Supercell\n")
    config_file.write("%10i%10i%10i\n" % (0, imcon, len(cell_coordinates)))
    config_file_index = 0

    [write_fastmc_style_unit_cell_vector_line([x, y, z]) for [x, y, z] in cell_vectors]
    [write_fastmc_style_atom_line(coordinate_line, config_file_index) for coordinate_line in
cell_coordinates]

    config_file.close()


def write_fastmc_control_file(system_temp, cutoff, delr, ewald_precision, equilibration_time,
simulation_length, guest_pp):

    control_file = open("CONTROL", 'w')
    partial_pressures = [guest_pp]
    fastmc_guest_section = "\n"
    for i in range(len(partial_pressures)):
        fastmc_guest_section += "\n&guest " + str(i + 1) + "\n pressure (bar) " +
str(partial_pressures[i]) + "\n&end\n"

    fastmc_equilibration_section = "\n\n#First do this number of steps (for each
node)\nequilibration " + equilibration_time\
                   + "\n\n# Simulation length and equilibration steps that count towards
average. (over all nodes)\nsteps " \
                   + simulation_length + "\nnumguests 100\n"


    fastmc_control_line = "GCMC\n\n# The state point\ntemperature " + system_temp +
fastmc_guest_section \
                   + "\n# specify cutoffs\ncutoff " + cutoff + "\ndelr " + delr + "\n# dealing
with coulombic forces\newald precision "\
                   + ewald_precision + fastmc_equilibration_section + "\nfinish"


    control_file.write(fastmc_control_line)
    control_file.close()


def write_fastmc_field_file(doap_job_name, unitcell_coordinates, charge_list, molecular_types,
fastmc_guest_file, vdw_list, supercell_dimensions):

    guest_field_data_list, guest_name, guest_vdw_data =
import_fastmc_guest_file(fastmc_guest_file)
    element_set = xyz_tools.create_atom_label_set(guest_field_data_list + unitcell_coordinates)

    field_file=open('FIELD', 'w')
    field_file.write("Simulation of " + doap_job_name + " supercell\n")
    field_file.write("UNITS kcal\n")
    field_file.write("molecular types " + str(molecular_types) + "\n")

    def write_fast_guest_field_atom_line(field_data_line):
        [atom_symbol, weight, charge, x, y, z] = field_data_line
        field_file.write("%8s".ljust(8) % (atom_symbol))
```

```
        field_file.write("%12.4f%12.5f%12.7f%12.7f%12.7f\n" % (float(weight), float(charge),
float(x), float(y), float(z)))

    def write_fast_guest_field_section(guest_name, guest_field_data_list):
        field_file.write("&guest " + guest_name + "\nNUMMOLS 0\nATOMS " +
str(len(guest_field_data_list)) + "\n")
        [write_fast_guest_field_atom_line(guest_field_data_line) for guest_field_data_line in
guest_field_data_list]
        field_file.write("finish\n")

    write_fast_guest_field_section(guest_name, guest_field_data_list)

    def write_fast_framework_field_section(doap_job_name, supercell_dimensions,
unitcell_coordinates, charge_list):
        field_file.write("Frozen " + doap_job_name + " framework\nNUMMOLS " +
str(product(supercell_dimensions)) + "\nATOMS " + str(len(unitcell_coordinates)) + "\n")
        for i in range(len(unitcell_coordinates)):
            atom = unitcell_coordinates[i][0]
            field_file.write("%8s%13.4f%9.4f 1 1\n" % (atom, MOLAR_MASSES[atom],
float(charge_list[i][1])))
        field_file.write("finish\n")

    write_fast_framework_field_section(doap_job_name, supercell_dimensions,
unitcell_coordinates, charge_list)

    def write_field_vdw_section(element_set, vdw_list):

        field_file.write("VDW")
        field_file.write(str(((len(element_set)**2) + len(element_set))/2).rjust(4) + "\n")

        def mix_sigma(sigma1, sigma2):
            return (sigma1 + sigma2) / 2.0

        def mix_epsilon(epsilon1, epsilon2):
            return (epsilon1 * epsilon2) ** (0.5)

        def write_vdw_field_line(atom1, atom2, sigma, epsilon):
            field_file.write(atom1.ljust(8) + atom2.ljust(8) + "lj %8.5f%8.5f\n" % (epsilon,
sigma))

        for i in range(len(element_set)):  #ie. [0,1,2,3,4] for a 5 element list
            for j in range(i+1):           #ie. [0,1,2,3] for range(3+1)
                atom1 = element_set[i]
                atom2 = element_set[j]
                mixed_sigma = mix_sigma(vdw_list[atom1][SIGMA], vdw_list[atom2][SIGMA])
                mixed_epsilon = mix_epsilon(vdw_list[atom1][EPSILON], vdw_list[atom2][EPSILON])
                write_vdw_field_line(atom1, atom2, mixed_sigma, mixed_epsilon)

        field_file.write("close")

    write_field_vdw_section(element_set, vdw_list)

    field_file.close()




def write_fastmc_simulation_files(doap_job_name, config_dictionary, charge_list, vdw_list,
unitcell_coordinates, supercell_coordinates, supercell_vectors):

    system_temp = str(int(config_dictionary["general_system_temp"]))
    cutoff = str(config_dictionary["gcmc_cutoff"])
    delr = str(config_dictionary["gcmc_delr"])
    ewald_precision = str(config_dictionary["gcmc_ewald_precision"])
    mol_types = config_dictionary["gcmc_mol_types"]
    imcon = int(config_dictionary["gcmc_imcon"])
    guest_pp = config_dictionary["fastmc_guest_pp"]
    simulation_length = config_dictionary["fastmc_sim_length"]
    equilibration_time = config_dictionary["fastmc_eq_time"]
```

```
    fastmc_guest_file = config_dictionary["gcmc_guest_file"]

    supercell_dimensions = map(int, [config_dictionary["gcmc_supercell_dim_a"],
config_dictionary["gcmc_supercell_dim_b"], config_dictionary["gcmc_supercell_dim_c"]])
    molecular_types = 2

    write_fastmc_config_file(doap_job_name, supercell_dimensions, supercell_vectors,
supercell_coordinates, imcon)
    write_fastmc_control_file(system_temp, cutoff, delr, ewald_precision, equilibration_time,
simulation_length, guest_pp)
    write_fastmc_field_file(doap_job_name, unitcell_coordinates, charge_list, molecular_types,
fastmc_guest_file, vdw_list, supercell_dimensions)
```

gauss_tools.py

```
import doap_logging

log = doap_logging.getLogger(__name__)



def print_gauss_file(directory, unit_cell_vectors, ordered_coordinates, filename):
    file_path = directory + filename + ".gjf"
    outfile = open(file_path, 'w')

    gauss_file_header = "# opt hf/3-21g \n\n\nTitle\n\n0 1\n"
    outfile.write(gauss_file_header)

    def write_gauss_vector_line(unit_cell_vector_coordinate_list):
        x,y,z = unit_cell_vector_coordinate_list
        outfile.write("TV  %15.5f  %15.5f  %15.5f\n" % (x, y, z))

    [write_gauss_vector_line([x, y, z]) for [CV, x, y, z] in unit_cell_vectors]

    def write_gauss_atom_line(atom_vector_coordinate_list):
        atom_symb, x, y, z = atom_vector_coordinate_list
        outfile.write("%3s  %15.5f  %15.5f  %15.5f\n" % (atom_symbol, float(x), float(y),
float(z)))

    [write_gauss_atom_line( [atom_symbol, x, y, z] ) for [atom_symbol, x, y, z] in
ordered_coordinates]

    outfile.write ("\n\n")
    outfile.close()
    log.info("Gaussian input file written to: " + filename)
```

get_gas_uptake.py

```
#!/usr/bin/env python

import sys
from queue_tools import CpmdJob
from queue_tools import RepeatJob
from queue_tools import FastmcJob
import doap_tools
import repeat_tools
import fastmc_tools
import molecular_mechanics_tools
import doap_logging
import xyz_tools
import cpmd_tools
import data_validation

log = doap_logging.getLogger(__name__)


def main(argv):
    if len(argv) != 3:
```

```
        return 1

    xyz_input_file_name = argv[1]
    doap_job_name = argv[2]

    config_dictionary = doap_tools.return_effective_config_dictionary("DOAP_OPTIONS")
    data_validation.validate_config_arguments(config_dictionary)
    data_validation.validate_config_options(config_dictionary, xyz_input_file_name)

    vdw_file_name = config_dictionary["general_vdw_file"]

    cpmd_number_of_processors = config_dictionary["cpmd_num_processors"]
    cpmd_tools.create_cpmd_input_file(xyz_input_file_name, doap_job_name, config_dictionary)
    cpmd_job = CpmdJob(doap_job_name, cpmd_number_of_processors)
    cpmd_job.run()
    repeat_tools.get_repeat_input_files(doap_job_name)
    repeat_tools.write_repeat_parameter_file(config_dictionary)
    repeat_job = RepeatJob(doap_job_name)
    repeat_job.run()

    charge_list = repeat_tools.import_repeat_charge_file("REPEAT_ESP.esp_fit.out",
config_dictionary)
    system_charge = float(config_dictionary["general_system_charge"])
    net_correct_charges =
molecular_mechanics_tools.adjust_charges_to_match_net_charge(charge_list, system_charge ,
0.0001)

    vdw_list = molecular_mechanics_tools.import_vdw_values(vdw_file_name)
    atom_coordinates = cpmd_tools.parse_cpmd_geometry_file("GEOMETRY.xyz")
    properly_labeled_atom_coordinates = xyz_tools.reassign_proper_atom_labels(atom_coordinates)

    order_list = xyz_tools.import_order_list(doap_job_name+".atom_order")

    ordered_charges = molecular_mechanics_tools.reorder_charges(order_list, charge_list)

    ordered_unit_cell_coordinates = xyz_tools.reorder_coordinates(order_list,
properly_labeled_atom_coordinates)

    unitcell_vectors = cpmd_tools.import_unit_vectors_from_cpmd_input_file(doap_job_name+".in")

    supercell_dimensions = map(int, [config_dictionary["gcmc_supercell_dim_a"],
config_dictionary["gcmc_supercell_dim_b"], config_dictionary["gcmc_supercell_dim_c"]])
    supercell_coordinates, supercell_vectors =
xyz_tools.create_supercell(ordered_unit_cell_coordinates, unitcell_vectors,
supercell_dimensions)

    fastmc_tools.write_fastmc_simulation_files(doap_job_name, config_dictionary, charge_list,
vdw_list, ordered_unit_cell_coordinates, supercell_coordinates, supercell_vectors)
    fastmc_number_of_processors = config_dictionary["fastmc_num_processors"]
    fastmc_job = FastmcJob(doap_job_name, fastmc_number_of_processors)
    fastmc_job.run()
    return 0


if __name__ == "__main__":
    try:
        log.info("Starting get_gas_uptake")
        sys.exit(main(sys.argv))

    except SystemExit:
        pass

    except Exception, e:
        log.exception(e)

    log.info("Shutting down get_gas_uptake")
```

GUEST

```
Carbon dioxide
NUMMOLS 1
ATOMS  3
Cx           12.011     +0.6645
Ox           15.999     -0.33225
Ox           15.999     -0.33225
rigid  1
    3    1    2    3
finish
Cx   -1.109708933         2.108201342         5.173087695
Ox   -2.119091025         1.530624172         5.162069333
Ox   -0.1003268405        2.685778512         5.184106058

VDW Cx        2.745   0.05948
VDW Ox        3.017   0.17023
```

INPUT_ZIF8

```
CV 16.991   00.000   00.000
CV 00.000   16.991   00.000
CV 00.000   00.000   16.991

Zn          0.0000000           4.2480000           8.4950000
C           1.4530000          10.0910000           6.9000000
Zn          4.2480000           0.0000000           8.4950000
C           0.1340000           6.4060000          10.5850000
H           2.0780000           6.0980000          13.1950000
C           9.9480000           1.5950000          15.3960000
N          15.4570000           9.0340000          13.8850000
H          14.5940000           6.4180000          12.2910000
H          15.2260000           6.3270000           7.6030000
H           2.3970000           4.7000000           6.4180000
H           2.0780000          13.1950000           6.0980000
H          14.5940000          10.5730000           4.7000000
C          11.6810000           1.7140000          10.7300000
C           5.3100000          15.2770000          10.7300000
N           0.5380000          11.6010000          10.0290000
C           3.1860000           6.7820000          14.7570000
C           2.0890000           8.3610000           2.0890000
H           2.0860000          10.0960000           6.1780000
H          12.2910000          14.5940000           6.4180000
H           1.2620000          16.2890000           9.8510000
H          14.9130000           6.0980000           3.7960000
H          14.6730000           6.4090000          15.3900000
C          10.5850000           6.4060000           0.1340000
C          13.8050000           2.2340000           6.7820000
N          11.6010000           0.5380000          10.0290000
N           5.3900000          10.0290000          16.4530000
N           1.5340000           7.9570000          13.8850000
N           9.0340000          15.4570000          13.8850000
H          10.5730000           2.3970000          12.2910000
C          10.5850000          10.5850000          16.8570000
H           6.3270000           7.6030000          15.2260000
C           0.1340000          10.5850000           6.4060000
H           9.1970000           1.3560000           7.2330000
N          13.8850000          15.4570000           9.0340000
H          16.2890000           9.8510000           1.2620000
H          16.0990000           2.1680000          10.2610000
C           6.2610000           1.7140000           5.3100000
N           5.3900000           0.5380000           6.9620000
C           6.2610000          15.2770000          11.6810000
C           1.4530000           6.9000000          10.0910000
H           4.7000000           2.3970000           6.4180000
C           1.5950000           1.5950000           7.0430000
Zn          8.4950000           0.0000000           4.2480000
H           1.6010000          10.5820000          14.6730000
```

H	7.6030000	10.6640000	1.7650000
H	10.8930000	14.9130000	13.1950000
C	14.9020000	8.6300000	2.0890000
H	10.8130000	6.8950000	2.0860000
C	10.7300000	1.7140000	11.6810000
N	1.5340000	9.0340000	3.1060000
C	5.3100000	10.7300000	15.2770000
H	15.7290000	7.1400000	16.2890000
C	6.9000000	10.0910000	1.4530000
N	7.9570000	15.4570000	3.1060000
C	8.6300000	2.0890000	14.9020000
H	6.0980000	2.0780000	13.1950000
C	14.7570000	13.8050000	10.2090000
H	2.1680000	0.8920000	6.7300000
H	2.3970000	10.5730000	12.2910000
H	9.8510000	15.7290000	0.7020000
C	10.0910000	6.9000000	1.4530000
H	6.3270000	15.2260000	7.6030000
H	14.9130000	3.7960000	6.0980000
H	9.7580000	9.1970000	15.6350000
H	6.0980000	3.7960000	14.9130000
H	10.8930000	3.7960000	2.0780000
C	15.2770000	5.3100000	10.7300000
H	10.5820000	2.3180000	15.3900000
N	16.4530000	11.6010000	6.9620000
Zn	8.4950000	0.0000000	12.7430000
C	15.3960000	15.3960000	7.0430000
H	10.8130000	10.0960000	14.9050000
N	11.6010000	16.4530000	6.9620000
N	6.9620000	11.6010000	16.4530000
N	0.5380000	10.0290000	11.6010000
H	15.6350000	7.7940000	7.2330000
C	15.3960000	9.9480000	1.5950000
H	6.8950000	6.1780000	14.9050000
H	15.3900000	10.5820000	2.3180000
C	8.3610000	14.9020000	14.9020000
C	6.4060000	0.1340000	10.5850000
C	6.4060000	10.5850000	0.1340000
H	13.1950000	10.8930000	14.9130000
H	15.2260000	9.3880000	10.6640000
C	13.8050000	14.7570000	10.2090000
N	10.0290000	5.3900000	16.4530000
Zn	12.7430000	0.0000000	8.4950000
C	14.7570000	3.1860000	6.7820000
C	14.9020000	2.0890000	8.6300000
H	3.7960000	2.0780000	10.8930000
H	6.4180000	14.5940000	12.2910000
C	2.2340000	10.2090000	3.1860000
H	12.2910000	10.5730000	2.3970000
C	11.6810000	10.7300000	1.7140000
H	9.7580000	7.7940000	1.3560000
H	6.4180000	12.2910000	14.5940000
H	14.6730000	1.6010000	10.5820000
H	7.2330000	1.3560000	9.1970000
C	5.3100000	6.2610000	1.7140000
Zn	0.0000000	8.4950000	4.2480000
H	12.2910000	2.3970000	10.5730000
N	15.4570000	13.8850000	9.0340000
C	10.5850000	16.8570000	10.5850000
C	6.9000000	1.4530000	10.0910000
C	1.7140000	6.2610000	5.3100000
N	3.1060000	7.9570000	15.4570000
H	7.7940000	7.2330000	15.6350000
H	6.4180000	2.3970000	4.7000000
H	6.4180000	4.7000000	2.3970000
H	0.7020000	1.2620000	7.1400000
H	10.5820000	14.6730000	1.6010000
N	0.5380000	5.3900000	6.9620000
H	4.7000000	6.4180000	2.3970000
N	9.0340000	13.8850000	15.4570000

H	2.3970000	6.4180000	4.7000000
N	16.4530000	5.3900000	10.0290000
H	1.3560000	9.1970000	7.2330000
H	1.6010000	6.4090000	2.3180000
H	2.0780000	3.7960000	10.8930000
H	2.0780000	10.8930000	3.7960000
N	6.9620000	5.3900000	0.5380000
N	10.0290000	11.6010000	0.5380000
Zn	8.4950000	4.2480000	0.0000000
C	14.7570000	10.2090000	13.8050000
C	6.2610000	5.3100000	1.7140000
C	16.8570000	6.4060000	6.4060000
C	15.3960000	1.5950000	9.9480000
C	6.9000000	6.9000000	15.5380000
N	1.5340000	3.1060000	9.0340000
H	6.4090000	15.3900000	14.6730000
H	13.1950000	14.9130000	10.8930000
H	9.8510000	1.2620000	16.2890000
C	15.5380000	10.0910000	10.0910000
H	9.1970000	15.6350000	9.7580000
H	0.7020000	7.1400000	1.2620000
N	10.0290000	16.4530000	5.3900000
C	13.8050000	6.7820000	2.2340000
H	10.5730000	14.5940000	4.7000000
H	3.7960000	14.9130000	6.0980000
C	16.8570000	10.5850000	10.5850000
H	1.7650000	10.6640000	7.6030000
N	3.1060000	1.5340000	9.0340000
H	0.7020000	9.8510000	15.7290000
C	15.5380000	6.9000000	6.9000000
N	6.9620000	0.5380000	5.3900000
Zn	0.0000000	8.4950000	12.7430000
C	10.7300000	5.3100000	15.2770000
C	15.2770000	10.7300000	5.3100000
H	10.0960000	14.9050000	10.8130000
Zn	4.2480000	8.4950000	0.0000000
N	16.4530000	10.0290000	5.3900000
C	15.2770000	6.2610000	11.6810000
H	14.9130000	13.1950000	10.8930000
N	10.0290000	0.5380000	11.6010000
H	16.0990000	14.8230000	6.7300000
N	7.9570000	1.5340000	13.8850000
C	13.8050000	10.2090000	14.7570000
H	10.5730000	12.2910000	2.3970000
N	7.9570000	3.1060000	15.4570000
H	14.8230000	10.2610000	0.8920000
H	7.2330000	9.1970000	1.3560000
H	10.8930000	13.1950000	14.9130000
H	6.0980000	14.9130000	3.7960000
Zn	12.7430000	8.4950000	0.0000000
C	1.7140000	11.6810000	10.7300000
C	2.0890000	2.0890000	8.3610000
C	2.0890000	8.6300000	14.9020000
N	7.9570000	13.8850000	1.5340000
C	10.2090000	2.2340000	3.1860000
H	6.4090000	1.6010000	2.3180000
C	10.2090000	13.8050000	14.7570000
C	1.7140000	5.3100000	6.2610000
Zn	8.4950000	12.7430000	0.0000000
C	3.1860000	14.7570000	6.7820000
H	15.6350000	9.7580000	9.1970000
N	13.8850000	7.9570000	1.5340000
H	10.5730000	4.7000000	14.5940000
C	3.1860000	2.2340000	10.2090000
N	5.3900000	6.9620000	0.5380000
C	14.7570000	6.7820000	3.1860000
H	1.6010000	2.3180000	6.4090000
C	6.7820000	13.8050000	2.2340000
H	7.7940000	15.6350000	7.2330000
H	2.3970000	12.2910000	10.5730000

N	3.1060000	15.4570000	7.9570000
H	9.3880000	6.3270000	1.7650000
H	13.1950000	2.0780000	6.0980000
C	6.9000000	15.5380000	6.9000000
H	14.5940000	4.7000000	10.5730000
N	15.4570000	3.1060000	7.9570000
N	15.4570000	7.9570000	3.1060000
C	5.3100000	1.7140000	6.2610000
H	7.6030000	1.7650000	10.6640000
H	10.2610000	0.8920000	14.8230000
H	6.0980000	13.1950000	2.0780000
H	1.7650000	6.3270000	9.3880000
C	6.7820000	2.2340000	13.8050000
C	6.4060000	16.8570000	6.4060000
N	13.8850000	9.0340000	15.4570000
C	10.5850000	0.1340000	6.4060000
N	9.0340000	3.1060000	1.5340000
N	11.6010000	6.9620000	16.4530000
C	10.0910000	1.4530000	6.9000000
C	10.7300000	11.6810000	1.7140000
Zn	0.0000000	12.7430000	8.4950000
C	14.9020000	8.3610000	14.9020000
H	0.8920000	14.8230000	10.2610000
C	1.5950000	7.0430000	1.5950000
N	1.5340000	13.8850000	7.9570000
C	7.0430000	15.3960000	15.3960000
C	6.7820000	14.7570000	3.1860000
C	10.2090000	14.7570000	13.8050000
H	14.9050000	10.8130000	10.0960000
C	7.0430000	1.5950000	1.5950000
H	10.0960000	2.0860000	6.1780000
H	14.9050000	6.8950000	6.1780000
H	3.7960000	10.8930000	2.0780000
H	1.3560000	7.7940000	9.7580000
H	14.9130000	10.8930000	13.1950000
N	9.0340000	1.5340000	3.1060000
H	16.0990000	6.7300000	14.8230000
H	10.2610000	16.0990000	2.1680000
C	15.3960000	7.0430000	15.3960000
H	10.6640000	1.7650000	7.6030000
C	6.2610000	11.6810000	15.2770000
N	5.3900000	16.4530000	10.0290000
H	12.2910000	6.4180000	14.5940000
C	10.0910000	10.0910000	15.5380000
H	2.1680000	10.2610000	16.0990000
H	7.1400000	16.2890000	15.7290000
C	2.2340000	13.8050000	6.7820000
C	2.2340000	3.1860000	10.2090000
C	1.5950000	15.3960000	9.9480000
C	2.2340000	6.7820000	13.8050000
H	14.5940000	12.2910000	6.4180000
C	10.0910000	15.5380000	10.0910000
H	4.7000000	10.5730000	14.5940000
N	13.8850000	1.5340000	7.9570000
C	11.6810000	6.2610000	15.2770000
H	9.3880000	10.6640000	15.2260000
H	6.8950000	14.9050000	6.1780000
H	6.1780000	2.0860000	10.0960000
N	6.9620000	16.4530000	11.6010000
H	15.7290000	0.7020000	9.8510000
C	11.6810000	15.2770000	6.2610000
C	10.2090000	3.1860000	2.2340000
H	2.1680000	6.7300000	0.8920000
H	2.3180000	15.3900000	10.5820000
C	14.9020000	14.9020000	8.3610000
C	1.5950000	9.9480000	15.3960000
C	9.9480000	15.3960000	1.5950000
C	6.7820000	3.1860000	14.7570000
C	1.7140000	10.7300000	11.6810000
H	10.8930000	2.0780000	3.7960000

```
H          14.6730000          15.3900000           6.4090000
C           8.3610000           2.0890000           2.0890000
H           6.7300000           2.1680000           0.8920000
H          10.6640000          15.2260000           9.3880000
C           3.1860000          10.2090000           2.2340000
N          11.6010000          10.0290000           0.5380000
C           6.4060000           6.4060000          16.8570000
H          15.7290000          16.2890000           7.1400000
H           6.1780000          10.0960000           2.0860000
C          10.7300000          15.2770000           5.3100000
N          16.4530000           6.9620000          11.6010000
C           2.0890000          14.9020000           8.6300000
C           8.6300000          14.9020000           2.0890000
N           3.1060000           9.0340000           1.5340000
N           0.5380000           6.9620000           5.3900000
H           3.7960000           6.0980000          14.9130000
C          15.2770000          11.6810000           6.2610000
H           6.7300000          14.8230000          16.0990000
H           4.7000000          14.5940000          10.5730000
H           7.1400000           0.7020000           1.2620000
H           2.0860000           6.8950000          10.8130000
H          13.1950000           6.0980000           2.0780000
```

molecular_mechanics_tools.py

```
import numpy
import doap_logging

log = doap_logging.getLogger(__name__)

SIGMA = 0
EPSILON = 1



def adjust_charges_to_match_net_charge(charge_list, desired_charge, adjust_by):
    new_charges = charge_list[:]
    net_charge = sum([charge for [atom,charge] in new_charges])
    adjustment = adjust_by*numpy.sign(desired_charge - net_charge)
    number_of_atoms_to_adjust=abs(int(numpy.round((desired_charge-net_charge)/adjustment)))
    if net_charge != charge:
        log.info("WARNING - total system charge of "+ str(net_charge))
        log.info("adjusting the first " + str(abs(number_of_atoms_to_adjust))+" atom charges by
"+str(adjustment))
    natoms = len(new_charges)
    if natoms < number_of_atoms_to_adjust:
        full_increm_cycles = int(number_or_atoms_to_adjust)/int(natoms)
        last_increm_cycle = int(((float(number_of_atoms_to_adjust)/natoms) -
int((float(number_of_atoms_to_adjust)/natoms)))*natoms)
        for j in range(full_increm_cycles):
            for i in range(abs(number_of_atoms_to_adjust)):
                new_charges[i][1]=new_charges[i][1]+adjustment
        for j in range(last_increm_cycle):
            for i in range(abs(number_of_atoms_to_adjust)):
                new_charges[i][1]=new_charges[i][1]+adjustment
    else:
       for i in range(abs(number_of_atoms_to_adjust)):
            new_charges[i][1]=new_charges[i][1]+adjustment
    return new_charges

def reorder_charges(order_list,charge_list):
    ordered_charges = zip(order_list,charge_list)
    ordered_charges.sort()
    ordered_charges = [[atom_symbol,charge] for (order,[atom_symbol,charge]) in ordered_charges]
    return ordered_charges

def import_vdw_values(filename):
    try:
```

```
        vdw_file=open(filename,'r')
        vdw_lines = vdw_file.readlines()
        vdw_file.close()
    except IOError,e:
        raise e
    vdw_dict = dict([[line.split()[0],map(float,line.split()[1:])] for line in vdw_lines])
    return vdw_dict
```

queue_tools.py

```
import os
import time
import doap_logging
import commands

log = doap_logging.getLogger(__name__)

class QstatJobReturnException(Exception):
    pass


class NullJob(object):

    def __init__(self, general_input_name, number_of_processors):
        self.general_input_name = general_input_name
        self.number_of_processors = number_of_processors

    def _is_title_line(self,line):
        return line.split() == ['Job', 'id', 'Name', 'User', 'Time', 'Use', 'S', 'Queue']

    def _try_get_queue_list(self):
        list_of_queue_entries = commands.getoutput('qstat').split("\n")
        first_line = list_of_queue_entries[0]
        if not self._is_title_line(first_line):
            raise QstatJobReturnException()
        else:
            return list_of_queue_entries

    def _get_queue_list(self):
        while True:
            try:
                return self._try_get_queue_list()
            except:
                log.info("Problem with qstat. Not returning jobs.  Waiting . . .")
                time.sleep(1)

    def _exists_on_queue(self):
        list_of_queue_entries = self._get_queue_list()
        del list_of_queue_entries[:2]             #removes title lines of qstat output
        list_of_job_ids = [entry.split(".")[0] for entry in list_of_queue_entries]
        return str(self.job_id) in str(list_of_job_ids)

    def _wait_for_completion(self):
        while(self._exists_on_queue()):
            time.sleep(10)

    def run(self):
        raise NotImplemented



class CpmdJob(object):

    def __init__(self, cpmd_input_name, number_of_processors):
        #prefer composition over inheritence (see gang of 4 book : design patterns)
        self.job = NullJob(cpmd_input_name, number_of_processors)

    def run(self):
```

```
        cpmdsubmit_output = commands.getoutput('cpmdsubmit '+self.job.general_input_name + ' ' +
self.job.number_of_processors)
        self.job.job_id = cpmdsubmit_output.split(".")[0]
        print "Job ID: "+self.job.job_id
        self.job._wait_for_completion()


class DlPolyJob(object):
    def __init__(self, dlpoly_input_name, number_of_processors):
        #prefer composition over inheritence (see gang of 4 book : design patterns)
        self.job = NullJob(dlpoly_input_name, number_of_processors)

    def run(self):
        dlpolysubmit_output = commands.getoutput('dlpoly2submit '+self.job.general_input_name +
' ' + self.job.number_of_processors)
        self.job.job_id = generalsubmit_output.split(".")[0]   ###CHECK THAT THIS IS TRUE
        self.job._wait_for_completion()


class FastmcJob(object):
    def __init__(self, fastmc_job_name, number_of_processors):
        #prefer composition over inheritence (see gang of 4 book : design patterns)
        self.job = NullJob(fastmc_job_name, number_of_processors)

    def run(self):
        fastmcsubmit_output = commands.getoutput('fastmcsubmit '+self.job.general_input_name + '
' + self.job.number_of_processors)
        self.job.job_id = fastmcsubmit_output.split(".")[0]
        self.job._wait_for_completion()


class RepeatJob(object):
    def __init__(self, repeat_input_name):
        #prefer composition over inheritence (see gang of 4 book : design patterns)
        self.job = NullJob(repeat_input_name, 1)

    def run(self):
        repeatsubmit_output = commands.getoutput('repeatsubmit REPEAT_ESP.cube')
        self.job.job_id = repeatsubmit_output.split(".")[0]
        self.job._wait_for_completion()
```

repeat_tools.py

```
import os
import sys
import doap_logging
from chem_constants import ATOMIC_NUMBER

log = doap_logging.getLogger(__name__)



username = os.getlogin()

def all(iterable):
    for element in iterable:
        if not element:
            return False
    return True


def get_repeat_input_files(job_name):
    if(os.path.exists('/shared_scratch/' + username + '/' + job_name + '/ELPOT')):
        log.info("ELPOT file found.\nCopying ELPOT file to local directory.\n")
        os.system('cp /shared_scratch/' + username + '/' + job_name + '/ELPOT ./')

        log.info("Copying GEOMETRY file to local directory.\n")
        os.system('cp /shared_scratch/' + username + '/' + job_name + '/GEOMETRY.xyz ./')
```

```
        log.info("Done Copying files to local directory.\nConverting ELPOT to cube file
ELPOT.cube")
        os.system('cpmd2cube.x ELPOT')

        log.info("ELPOT.cube file finished renaming to REPEAT_ESP.cube.\n")
        os.system('mv ELPOT.cube REPEAT_ESP.cube')

    else:
        log.error("--CPMD data incomplete and/or ELPOT file in " + '/shared_scratch/' + username
+ '/' + job_name + '/ not found')
        raise Exception("CPMD data incomplete and/or ELPOT file not found.")


def write_repeat_parameter_file(repeat_config_dictionary):

    repeat_periodicity = str(repeat_config_dictionary["repeat_periodicity"])
    repeat_vdw_factor = str(repeat_config_dictionary["repeat_vdw_factor"])
    repeat_resp_penalties = str(repeat_config_dictionary["repeat_resp_penalties"])
    repeat_read_cutoff = str(repeat_config_dictionary["repeat_read_cutoff"])
    repeat_r_cutoff = str(repeat_config_dictionary["repeat_r_cutoff"])
    repeat_apply_symmetry_restrain =
str(repeat_config_dictionary["repeat_apply_symmetry_restrain"])
    repeat_use_goddard = str(repeat_config_dictionary["repeat_use_goddard"])
    repeat_goddard_weight = str(repeat_config_dictionary["repeat_goddard_weight"])
    repeat_charge = str(repeat_config_dictionary["general_system_charge"])

    repeat_input_line = "Input ESP file name in cube format\nREPEAT_ESP.cube\nFit molecular(0)
or periodic(1:default) system?\n"\
                        + repeat_periodicity + "\nvan der Waals scaling factor (default = 1.0)\n"
+ repeat_vdw_factor\
                        + "\nApply RESP penalties?, no(0:default), yes(1)\n" +
repeat_resp_penalties + "\nRead cutoff radius? no(0), yes(1:default)\n"\
                        + repeat_read_cutoff + "\nIf flag above=1 provide R_cutoff next (in
Bohrs)\n"\
                        + repeat_r_cutoff + "\nApply symmetry restrain? no(0:default), yes(1)\n" +
repeat_apply_symmetry_restrain\
                        + "\nUse Goddard-type restrains?no(0:default), yes(1)\n" +
repeat_use_goddard + "\nIf flag above=1 then provide weight next\n"\
                        + repeat_goddard_weight + "\nEnter total charge of the system\n" +
repeat_charge + "\n"

    if os.path.exists("./REPEAT_param.inp"):
        os.system("mv ./REPEAT_param.inp ./REPEAT_param.bak")
        log.info("REPEAT params found in current directory, backed up as REPEAT_param.bak")

    repeat_params=open("REPEAT_param.inp", "w")
    repeat_params.write(repeat_input_line)
    repeat_params.close()

def is_repeat_charge_line(line):
    return len(line.split()) > 0 and line.split()[0].upper() == "CHARGE"

def parse_charges_from_repeat_output(repeat_output_lines):
    def parse_out_atom_and_charge(repeat_output_line):
        return [ATOMIC_NUMBER[int(repeat_output_line.split()[4])],
float('%9.4f'%float(repeat_output_line.split()[6]))]

    return [parse_out_atom_and_charge(line) for line in repeat_output_lines if
is_repeat_charge_line(line)]


def import_repeat_charge_file(file_name, charge_config_dictionary):
    try:
        repeat_charge_file = open(file_name,'r')
    except IOError,e:
        raise e

    repeat_output_lines = repeat_charge_file.readlines()
    repeat_charges = parse_charges_from_repeat_output(repeat_output_lines)
```

```
    return repeat_charges
```

submit_gas_uptake_job.py

```
#!/usr/bin/env python
import os, sys, optparse, time

debug=False
debug=True

username=os.getlogin()

# ------------------------------------------------------------------------
# DOAP (get_gas_uptake) queue submission script - python version
# The purpose of this script is to allow the DOAP gas occupancy
# determination framework to run on a PBS qeueing system.
#
# Sept 2011 - Dave van Rijswijk
# ------------------------------------------------------------------------

usage = "\nUsage: submit_gas_uptake_job [xyz input file] [job name]\n"

pid = str(os.getpid())

if len(sys.argv) != 3:
  print "\nToo many/few arguments."
  print usage
  sys.exit(0)


max_cpus = 1               #-max no. of CPUs that can be requested
mem_default="1gb"           #-default memory request to pass to qsub
default_queue=""            #-leave blank ("") to use Wooki default
                            # (not the same as node attributes)
shared_memory_only=False   #some codes only have shared memory parallel executables
n_cpus = 1
scratch_dir = "/shared_scratch/"+username
shared_memory_job = False

PACKAGE_DIR = os.getcwd()+"/"

#*****************************************************************************
# QSUB bash script Begin
#*****************************************************************************
line="""

export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/home/program/LIBRARIES/CMKL/8.1/lib/em64t
#set
cd $SCRATCH_DIR
mkdir \$JOBNAME
cd \$JOBNAME

cp $STARTDIR/$JOBINPUT ./
cp $PACKAGE_DIR* ./

touch $STARTDIR/$JOBNAME.log
ln -s $STARTDIR/$JOBNAME.log output

echo "Welcome to AUTO OCC">>output
echo "Job Name: " \$JOBNAME>>output
echo "Main Host:" \$HOSTNAME>>output
echo "PBS JOB ID: " \$PBS_JOBID>>output
echo "Job Start Date: " $DATETIME >> output
echo
"-------------------------------------------------------------------------------------------------------
-----------" >> output
```

```
# ==========================================
# SERIAL EXECUTION ONLY (PARALLEL NOT NEEDED)
# ==========================================

DOAP_EXE=$PACKAGE_DIRget_gas_uptake
\$DOAP_EXE $JOBINPUT $JOBNAME  >> output

cp -r ../$JOBNAME $STARTDIR

"""
#*****************************************************************************************
# QSUB bash script END
#*****************************************************************************************


def abort(message):
    print message
    print "*****No job submitted to queue*****"
    sys.exit()

if len(sys.argv) == 0:
   abort ("Missing argument:  Need to specify the job name")
else:
   jobname = sys.argv[2]
   if jobname[-3:]==".in":                          #Strip ".in" from argument
      jobname=jobname[:-3]

   jobinput = sys.argv[1]
   if not os.path.exists(jobinput):            #check if job input exists
      abort ("Missing input file: "+jobinput+" could not be found.")


# ---------------------------------------------------
# Make sure that user has a directory on /shared_scratch
# ---------------------------------------------------
if not os.path.exists("/shared_scratch/"+username):
   print "user "+username+ " does not have shared scratch diretory, creating.\n"
   os.popen("mkdir /shared_scratch/"+username)

#---------------------------------------------------
# Make replacements in the bash qsub script
#---------------------------------------------------
line=line.replace("$STARTDIR",os.getcwd())
line=line.replace("$SCRATCH_DIR",scratch_dir)
line=line.replace("$JOBNAME",jobname)
line=line.replace("$NODES",str(n_cpus))
line=line.replace("$JOBINPUT",jobinput)
line=line.replace("$DATETIME",str(time.ctime()))
line=line.replace("$PACKAGE_DIR",str(PACKAGE_DIR))

#---------------------------------------------------
# Construct PBS directives
#---------------------------------------------------
PBS_directives="""#PBS -m n
#PBS -o std.out
#PBS -j oe """


#-----------------------------
# Add jobname to PBS directive
#-----------------------------
PBS_directives="#PBS -N "+jobname+"\n"+PBS_directives


line = "qsub << eof \n"+PBS_directives+"\n"+line

os.system (line)
#print line
```

validation_criteria.py

```
#Name of columns in value lists of the criteria dictionary
INFO  = 0
REGEX = 1
TYPE  = 2

CRITERIA = {
"CPMD_NUM_PROCESSORS"            :["Number of Processors to use in CPMD","^[1-9]{1}$|^[0-9]+
$",int],
"CPMD_CONVERGENCE_CRIT_ORBITALS" :["Orbital Convergence criteria in CPMD","^[0-9]*\.?[0-9]+d[-
+]?[0-9]",str],
"CPMD_MAX_ITERATIONS"            :["Maximum number of optimization steps","^[0-9]+$",int],
"CPMD_PSP_PATH"                  :["Absolute path for pseudopotential files","^((?:\/[a-zA-Z0-
9]+(?:_[a-zA-Z0-9]+)*(?:\-[a-zA-Z0-9]+)*)+)\/?$",str],
"CPMD_FUNCTIONAL"        :["Default functional type","^[A-Za-z]+",str],
"CPMD_PSP_TYPE"          :["Default pseudopotential type","^[A-Za-z0-9]+$",str],
"CPMD_LMAX_FLAG"         :["Default highest orbital consideration","^[S,P,D,F]{1}",str],
"CPMD_CUTOFF"                    :["Cutoff for CPMD calculation in angstroms","^([0-
9]*|\d*\.\d{1}?\d*)$",float],
"CPMD_CHARGE"                    :["Charge of system for CPMD calculation","^[+-]?[0-9]+$",int],
"CPMD_OPTIMIZE_HYDROGENS"        :["Flag for whether or not to optimize Hydrogens","^True$|
^False$",bool],
"DLPOLY_GUEST_GRIDPOINTS"        :["number of gridlines per dimension for guest insertion
search","^([0-9]*|\d*\.\d{1}?\d*)$",float],
"DLPOLY_MIN_GUEST_DISTANCE"      :["Minimum distance to framework for guest insertion in
Angstroms.","^([0-9]*|\d*\.\d{1}?\d*)$",float],
"GCMC_MOL_TYPES"               :["Number of DL_POLY Molecular types","^[1-9]{1}$|^[0-9]{1}[1-9]
{1}$",int],
"GCMC_SUPERCELL_DIM_A"         :["Multiplicity of supercell in Direction A","^[1-9]{1}$|^[0-9]
{1}[1-9]{1}$",int],
"GCMC_SUPERCELL_DIM_B"         :["Multiplicity of supercell in Direction B","^[1-9]{1}$|^[0-9]
{1}[1-9]{1}$",int],
"GCMC_SUPERCELL_DIM_C"         :["Multiplicity of supercell in Direction C","^[1-9]{1}$|^[0-9]
{1}[1-9]{1}$",int],
"DLPOLY_LEVCFG"                  :["See DL_POLY manual","^[0-2]$",int],
"GCMC_IMCON"                   :["See DL_POLY manual","^[0-7]$",int],
"GENERAL_VDW_FILE"               :["Filename for VDW params","^\.{0,2}((?:\/?[a-zA-Z0-9]+
(?:_[a-zA-Z0-9\.]+)*(?:\-[a-zA-Z0-9\.]+)*)+)\/?$",str],
"GCMC_GUEST_FILE"              :["Filename for guest information","^\.{0,2}((?:\/?[a-zA-Z0-9]+
(?:_[a-zA-Z0-9\.]+)*(?:\-[a-zA-Z0-9\.]+)*)+)\/?$",str],
"GCMC_CUTOFF"                 :["Required forces cutoff (Angstroms)","^([0-9]*|\d*\.\d{1}?\d*)
$",float],
"GCMC_DELR"                    :["Verlet neighbour list shell width (Angstroms)","^([0-
9]*|\d*\.\d{1}?\d*)$",float],
"GCMC_EWALD_PRECISION"        :["Ewald sum for electrostatics, with automatic parameter
optimisation (0 < f < 1.E . 4)","^[0-9]*\.?[0-9]+d[-+]?[0-9]",str],
"REPEAT_ESP_CUBEFILE"            :["Input ESP file name in cube format","^[A-Za-z]{1}[A-Za-z0-
9_]*[\.]?[A-Za-z0-9\_]*$",str],
"REPEAT_PERIODICITY"             :["Fit molecular(0) or periodic(1:default) system?","^[0-
1]$",int],
"REPEAT_VDW_FACTOR"              :["van der Waals scaling factor (default = 1.0)","^[0-9]+[\.]
{1}[0-9]+$",float],
"REPEAT_RESP_PENALTIES"          :["Apply RESP penalties?, no(0:default), yes(1)","^[0-
1]$",int],
"REPEAT_READ_CUTOFF"             :["Read cutoff radius? no(0), yes(1:default)","^[0-1]$",int],
"REPEAT_R_CUTOFF"                :["If flag above=1 provide R_cutoff next (in Bohrs)","^[0-9]+
[\.]{1}[0-9]+$",float],
"REPEAT_APPLY_SYMMETRY_RESTRAIN" :["Apply symmetry restrain? no(0:default), yes(1)","^[0-
1]$",int],
"REPEAT_USE_GODDARD"             :["Use Goddard-type restrains? no(0:default), yes(1)","^[0-
1]$",int],
"REPEAT_GODDARD_WEIGHT"          :["If flag above=1 then provide weight next","^[0-9]+[\.]{1}[0-
9]+$",float],
"REPEAT_CHARGE"                  :["Enter total charge of the system","^[0-9]+[\.]{1}[0-9]+
$",float],
"GCMC_NUMBER_TO_AVERAGE"         :["See GCMC README","^[1-9]{1}$|^[0-9]+$",int ],
```

```
"GCMC_ACCEPTANCES"               :["See GCMC README","^[1-9]{1}$|^[0-9]+$",int ],
"GCMC_PRODUCTION_AVERAGE"        :["See GCMC README","^[1-9]{1}$|^[0-9]+$",int ],
"GCMC_IDEAL_GAS_PRESSURE"        :["See GCMC README","^[1-9]{1}$|^[0-9]+$",int ],
"GCMC_PROBABILITY_PLOT"          :["See GCMC README","^[1-9]{1}$|^[0-9]+$",int ],
"GCMC_WINDOW_SIZE"               :["Default window size for convergence criteria","^[1-9]{1}$|
^[0-9]+$",int],
"GCMC_CONVERGENCE_CRITERIA"      :["GCMC Convergence criteria","^[0-9]+[\.]{1}[0-9]+$",float],
"GCMC_GCMCPAR"                   :["parameters for GCMC submission script","^[A-Za-z1-9]+
$",str],
"GCMC_NUM_PROCESSORS"            :["Number of branches for GCMC run","^[1-9]{1}$|^[0-9]+$",int],
"GCMC_WAIT_TIME"                 :["Wait time between convergence assessments in minutes.","^[1-
9]{1}$|^[0-9]+$",int ],
"GCMC_GUEST_NAME"                :["Name of Guest to pull from library","^[a-zA-Z0-9]*$",str ],
"AO_CONTROL_USE"                 :["Backup, Overwrite, or Use Current CONTROL file", "^BACKUP$|
^OVERWRITE$|^USE$|^backup$|^overwrite$|^use$",str],
"GENERAL_SYSTEM_TEMP"           :["Temperature for CONTROL file","^[1-9]{1}$|^[0-9]+$",int],
"MAIN_GCMC_PACKAGE"              :["GCMC Package to use, DLPOLY or FASTGCMC","^DLPOLY$|
^FASTMC$",str],
"FASTMC_GUEST_PP"                :["Partial pressure of first guest in fastGCMC","^([0-
9]*|\d*\.\d{1}?\d*)$",float],
"FASTMC_NUM_GUESTS"              :["Number of guest types in fastGCMC","^[1-9]{1}$|^[0-9]+
$",int],
"FASTMC_SIM_LENGTH"                :["Simulation length for FASTGCMC (timesteps)","^[1-9]{1}$|
^[0-9]+$",int ],
"FASTMC_EQ_TIME"                   :["Simulation equilibration time for FASTGCMC (units)","^[1-
9]{1}$|^[0-9]+$",int ],
"GENERAL_SYSTEM_CHARGE"          :["Charge of system","^[+-]?[0-9]+$",int],
"MAIN_CHARGE_PACKAGE"            :["Charge package used","^CPMD$|^VASP$",str],
"MAIN_OPTION_PROFILE"            :["Option profile used for general arguments","^GENERAL$",str],
"MAIN_POINT_CHARGE_PACKAGE"      :["Main package for resolving point charges","^REPEAT$",str]
}
```

vanderwaals_Parameters

```
H         2.5711  0.0440
He        2.1043  0.0560
Li        2.1836  0.0250
Be        2.4455  0.0850
B         3.6375  0.1800
C         3.4309  0.1050
N         3.2607  0.0690
O         3.1181  0.0600
F         2.9970  0.0500
Ne        2.8892  0.0420
Na        2.6576  0.0300
Mg        2.6914  0.1110
Al        4.0082  0.5050
Si        3.8264  0.4020
P         3.6946  0.3050
S         3.5948  0.2740
Cl        3.5164  0.2270
Ar        3.4460  0.1850
K         3.3961  0.0350
Ca        3.0282  0.2380
Sc        2.9355  0.0190
Ti        2.8286  0.0170
V         2.8010  0.0160
Cr        2.6932  0.0150
Mn        2.6380  0.0130
Fe        2.5943  0.0130
Co        2.5587  0.0140
Ni        2.5248  0.0150
Cu        3.1137  0.0050
Zn        2.4616  0.1240
Ga        3.9048  0.4150
Ge        3.8130  0.3790
As        3.7685  0.3090
```

Se	3.7462	0.2910
Br	3.7320	0.2510
Kr	3.6892	0.2200
Rb	3.6652	0.0400
Sr	3.2438	0.2350
Y	2.9801	0.0720
Zr	2.7832	0.0690
Nb	2.8197	0.0590
Mo	2.7190	0.0560
Tc	2.6709	0.0480
Ru	2.6397	0.0560
Rh	2.6094	0.0530
Pd	2.5827	0.0480
Ag	2.8045	0.0360
Cd	2.5373	0.2280
In	3.9761	0.5990
Sn	3.9128	0.5670
Sb	3.9378	0.4490
Te	3.9823	0.3980
I	4.0090	0.3390
Xe	3.9235	0.3320
Cs	4.0242	0.0450
Ba	3.2990	0.3640
La	3.1377	0.0170
Ce	3.1680	0.0130
Pr	3.2126	0.0100
Nd	3.1850	0.0100
Pm	3.1600	0.0090
Sm	3.1360	0.0080
Eu	3.1119	0.0080
Gd	3.0005	0.0090
Tb	3.0745	0.0070
Dy	3.0540	0.0070
Ho	3.0371	0.0070
Er	3.0210	0.0070
Tm	3.0059	0.0060
Yb	2.9890	0.2280
Lu	3.2429	0.0410
Hf	2.7983	0.0720
Ta	2.8241	0.0810
W	2.7342	0.0670
Re	2.6317	0.0660
Os	2.7796	0.0370
Ir	2.5302	0.0730
Pt	2.4535	0.0800
Au	2.9337	0.0390
Hg	2.4099	0.3850
Tl	3.8727	0.6800
Pb	3.8282	0.6630
Bi	3.8932	0.5180
Po	4.1952	0.3250
At	4.2318	0.2840
Rn	4.2451	0.2480
Fr	4.3654	0.0500
Ra	3.2758	0.4040
Ac	3.0985	0.0330
Th	3.0255	0.0260
Pa	3.0504	0.0220
U	3.0246	0.0220
Np	3.0504	0.0190
Pu	3.0504	0.0160
Am	3.0121	0.0140
Cm	2.9631	0.0130
Bk	2.9747	0.0130
Cf	2.9515	0.0130
Es	2.9391	0.0120
Fm	2.9275	0.0120
Md	2.9168	0.0110
No	2.8936	0.0110
Lr	2.8829	0.0110

```
Cx        2.745   0.05948
Ox        3.017   0.17023
```

xyz_tools.py

```
#!/usr/bin/env python

import doap_logging
import unittest
import doap_tools
from chem_constants import ELEMENTS

log = doap_logging.getLogger(__name__)

def get_atom_symbols_from_coordinate_list(coordinate_list):
    return [atom_symbol for [atom_symbol, x, y, z] in coordinate_list]

def get_locations_from_coordinate_list(coordinate_list):
    return [(x,y,z) for [atom_symbol, x, y, z] in coordinate_list]


def find_proper_atom_labels(atom_label_list):
    try:
        return [max([atom for atom in ELEMENTS if user_atom.startswith(atom)]) for user_atom in
atom_label_list]
    except:
        return None

def reassign_proper_atom_labels(atom_coordinates):
    atom_labels = get_atom_symbols_from_coordinate_list(atom_coordinates)
    locations = get_locations_from_coordinate_list(atom_coordinates)
    proper_atom_labels = find_proper_atom_labels(atom_labels)
    return [(symbol, x, y, z) for (symbol, (x, y, z)) in zip(proper_atom_labels, locations)]


def get_four_columned_lines(line_list):
    return [[x for x in y.split()] for y in line_list if len(y.split()) == 4]

def cell_vector_filter(symbol):
    return symbol == "CV"

def atom_symbol_filter(symbol):
    return symbol != "CV"

def extract_lines_by_symbol_filter(filter, coordinate_lines):
    return [[symbol, float(x), float(y), float(z)] for [symbol, x, y, z] in coordinate_lines if
filter(symbol)]

def parse_unit_cell_file(filename):
    try:
       input_file=open(filename, 'r')
        line_list = input_file.readlines()
        input_file.close()
    except IOError, e:
        raise e
    four_columned_lines = get_four_columned_lines(line_list)
    atom_coordinates = extract_lines_by_symbol_filter(atom_symbol_filter, four_columned_lines)
    unit_cell_vectors = extract_lines_by_symbol_filter(cell_vector_filter, four_columned_lines)
    return (atom_coordinates, unit_cell_vectors)

def return_list_of_element_x(list_of_lists, n):
    return [list[n] for list in list_of_lists]

def create_atom_label_set(atom_lines):
    return list(set(return_list_of_element_x(atom_lines,0)))

def reorder_H_atoms_to_last(atom_label_set):
    temp_labels = list(atom_label_set)
```

```
    if "H" in temp_labels:
        temp_labels.remove("H") #atom_labels is a set.
        temp_labels.append("H")
    return temp_labels

def get_atom_quantities(atom_coordinates, atom_labels):
    return dict([(w, sum([1 for [symbol, x, y, z] in atom_coordinates if symbol == w])) for w in
atom_labels])

def index_and_sort_coordinate_list(atom_coordinates,atom_labels):
    indexed_coordinates = zip(atom_coordinates, range(len(atom_coordinates)))

    #retyping zipped tuples into lists
    indexed_coordinates = [[[atom_symbol, x, y, z], index] for ([atom_symbol, x, y, z], index)
in indexed_coordinates]

    atom_index = [[index for [[atom_symbol, x, y, z], index] in indexed_coordinates if
atom_symbol == atom] for atom in atom_labels]
    ordered_coordinates = [[ [atom_symbol, x, y, z] for [[atom_symbol, x, y, z], index] in
indexed_coordinates if atom_symbol == atom] for atom in atom_labels]

    atom_index = doap_tools.reduce_list_depth(atom_index)
    ordered_coordinates = doap_tools.reduce_list_depth(ordered_coordinates)

    return ordered_coordinates, atom_index

def hydrogens_present(atom_labels):
    return [1] == [1 for x in atom_labels if x.upper() == "H"]


def write_atom_order_to_file(index,order_file_name):
    ORDER_FILE = open(order_file_name, 'w')
    [ORDER_FILE.write(str(x) + "\n") for x in index]
    ORDER_FILE.close()


def import_order_list(order_file):
    try:
        order_file=open(order_file, 'r')
        order_lines = order_file.readlines()
        order_file.close()
    except IOError, e:
        raise e

    order_list = map(int, order_lines)
    return order_list

def reorder_coordinates(order_list, coordinates):
    ordered_coordinates = zip(order_list, coordinates)
    ordered_coordinates.sort()
    ordered_coordinates = [[atom_symbol, x, y, z] for (order, [atom_symbol, x, y, z]) in
ordered_coordinates]
    return ordered_coordinates

def is_floatable(value):
    try:
        float(value)
        return True
    except:
        return False

def is_a_coordinate_line(line):
    split_line = line.split()
    if not len(split_line) == 4:
        return False
    return min([split_line[0][0].isalpha()] + [is_floatable(coordinate) for coordinate in
split_line[1:]])

def vector_sum(vector1, vector2):
    [x1, y1, z1] = vector1
```

```
    [x2, y2, z2] = vector2
    return [(x2 + x1), (y2 + y1), (z2 + z1)]

def scale_vector(vector, factor):
    [x, y, z] = vector
    return [x * factor, y * factor, z * factor]

def create_supercell(unit_cell_coordinates, unit_cell_vectors, supercell_dimensions):
    [xmult, ymult, zmult] = supercell_dimensions
    [ucv1, ucv2, ucv3] = [[ucvx1, ucvy1, ucvz1], [ucvx2, ucvy2, ucvz2], [ucvx3, ucvy3, ucvz3]] =
unit_cell_vectors
    vector_mask = [[naa, nbb, ncc] for naa in range(xmult) for nbb in range(ymult) for ncc in
range(zmult)]
    translation_vectors = [[naa * ucvx1 + nbb * ucvx2 + ncc * ucvx3,
                            naa * ucvy1 + nbb * ucvy2 + ncc * ucvy3,
                            naa * ucvz1 + nbb * ucvz2 + ncc * ucvz3] for [naa,nbb,ncc] in
vector_mask]
    supercell_coordinates = [([atom_symbol] + vector_sum(translation_vector, [x, y, z])) for
translation_vector in translation_vectors for [atom_symbol, x, y, z] in unit_cell_coordinates]
    supercell_vectors = [scale_vector(ucv1, xmult), scale_vector(ucv2, ymult),
scale_vector(ucv3, zmult)]
    return supercell_coordinates, supercell_vectors
```

unit_tests/test_cpmd_tools.py

```
 import inspect
import unittest
import os
import sys

from unit_testing_constants import EXAMPLE_XYZ_INPUT
from unit_testing_constants import EXAMPLE_RESULTING_CPMD_INPUT_FILE
from unit_testing_constants import EXAMPLE_CPMD_CHARGE_CONFIG
from unit_testing_constants import EXAMPLE_CPMD_GEOMETRY_FILE
from unit_testing_constants import EXAMPLE_CPMD_GEOMETRY_IMPORT_RESULT


def fix_parent_path():
    dir_path = os.getcwd()
    parent_dir_path = os.path.abspath(dir_path + "/../")
    if parent_dir_path not in sys.path:
        sys.path.insert(0, parent_dir_path)

fix_parent_path()

import cpmd_tools



class IsBinaryPspFileTests(unittest.TestCase):

    def setUp():
        binary_file = open("example_unit_testing_binary_file",'w')

    def test_general_case(self):
        result = is_binary_psp_file(filename)
'''

class CountAllNonHydrogenAtomsTests(unittest.TestCase):

    def test_regular_case(self):
        atom_counts = {"H":4,"C":5,"He":6,"Zn":7}
        atom_labels = ["H","C","He","Zn"]
        result = cpmd_tools.count_all_non_hydrogen_atoms(atom_counts,atom_labels)
        self.assertEqual(result,18)

    def test_no_hydrogen_case(self):
        atom_counts = {"I":4,"C":5,"He":6,"Zn":7}
        atom_labels = ["I","C","He","Zn"]
```

```
        result = cpmd_tools.count_all_non_hydrogen_atoms(atom_counts,atom_labels)
        self.assertEqual(result,22)

class VdbPresentTests(unittest.TestCase):
    def test_general_case(self):
        psp_choices = ["VDB_PBE_C.psp", "VDB_PBE_H", "GSP_PBE_Zn.psp"]
        result = cpmd_tools.vdb_present(psp_choices)
        self.assertTrue(result)

    def test_no_VDB_present_case(self):
        psp_choices = ["PBE_C.psp", "PBE_H", "GSP_PBE_Zn.psp"]
        result = cpmd_tools.vdb_present(psp_choices)
        self.assertFalse(result)


class HydrogensPresentTests(unittest.TestCase):
    def test_hydrogens_present_case(self):
        atom_labels = ["H","C","Zn","L"]
        result = cpmd_tools.hydrogens_present(atom_labels)
        self.assertTrue(result)

    def test_no_hydrogens_present_case(self):
        atom_labels = ["K","C","Zn","L"]
        result = cpmd_tools.hydrogens_present(atom_labels)
        self.assertFalse(result)

    def test_hydrogen_like_atoms_present_case(self):
        atom_labels = ["He","C","Hu","Zn","L"]
        result = cpmd_tools.hydrogens_present(atom_labels)
        self.assertFalse(result)




class ImportUnitVectorsFromCpmdInputFile(unittest.TestCase):

    def setUp():
        f1 = open("unit_test_cpmd_input",'w')
        [f1.write(x) for x in EXAMPLE_RESULTING_CPMD_INPUT_FILE]
        f1.close()


    def test_general_case(self):
        result = cpmd_tools.import_unit_vectors_from_cpmd_input_file("unit_test_cpmd_input")
        self.assertEqual(result, [[16.991, 0.0, 0.0], [0.0, 16.991, 0.0], [0.0, 0.0, 16.991]])

    def tearDown():
        os.system("rm unit_test_cpmd_input")



class ParseCpmdGeometryFileTests(unittest.TestCase):

    def setUp():
        f1 = open("example_cpmd_geometry_file",'w')
        [f1.write(x) for x in EXAMPLE_CPMD_GEOMETRY_FILE]
        f1.close()


    def test_general_case(self):
        result = cpmd_tools.parse_cpmd_geometry_file("example_cpmd_geometry_file")
        self.assertEqual(result, EXAMPLE_CPMD_GEOMETRY_IMPORT_RESULT)

    def tearDown():
        os.system("rm example_cpmd_geometry_file")
```

```
class CreateCpmdInputFileTests(unittest.TestCase):

    def setUp():
        f1 = open("unit_test_xyz_input",'w')
        [f1.write(x) for x in EXAMPLE_XYZ_INPUT]
        f1.close()


        xyz_input_file_name = "unit_test_xyz_input"
        cpmd_job_name = "unit_test_job"
        charge_config = EXAMPLE_CPMD_CHARGE_CONFIG

        cpmd_tools.create_cpmd_input_file(xyz_input_file_name, cpmd_job_name, charge_config)


    def test_general_case(self):
        f1 = open("unit_test_xyz_input",'r')
        result = f1.readlines()
        f1.close()
        self.assertEqual(result, EXAMPLE_XYZ_INPUT_RESULT)

    def tearDown():
        os.system("rm unit_test_xyz_input")
```

unit_tests/test_doap_tools.py

```
import inspect
import unittest
import os
import sys
from unit_testing_constants import EXAMPLE_JOB_OPTION_LIST
from unit_testing_constants import EXAMPLE_DOAP_OPTIONS_CONFIG_DICT_RESULT



def fix_parent_path():
    dir_path = os.getcwd()
    parent_dir_path = os.path.abspath(dir_path + "/../")
    if parent_dir_path not in sys.path:
        sys.path.insert(0, parent_dir_path)

fix_parent_path()

import doap_tools


class RemoveCommentFromLineListTests(unittest.TestCase):
    def test_general_case(self):
        line_list = ["line1#comment1\n","line2#comment2\n"]
        result = doap_tools.remove_comments_from_line_list(line_list)
        sefl.assertEqaul(result,['line1', 'line2'])

class GetMainOptionPrefixesTests(unittest.TestCase):
    def test_general_case(self):
        result = get_main_option_prefixes(uncommented_job_config_line_list)


class ReturnEffectiveConfigDictionaryTests(unittest.TestCase):
    def setUp():
        f1 = open("unit_testing_example_doap_options_file",'w')
        [f1.write(x) for x in EXAMPLE_JOB_OPTION_LIST]
        f1.close()

    def test_general_case(self):
```

```
        result =
doap_tools.return_effective_config_dictionary("unit_testing_example_doap_options_file")
        self.assertEqaul(result, EXAMPLE_DOAP_OPTIONS_CONFIG_DICT_RESULT)

    def tearDown():
        os.system("rm unit_testing_example_doap_options_file")
```

unit_tests/test_fastmc_tools.py

```
import inspect
import unittest
import os
import sys

from unit_testing_constants import GUEST_FILE_EXAMPLE
from unit_testing_constants import GUEST_FILE_COMPONENT_RESULTS
from unit_testing_constants import GUEST_FILE_IMPORT_RESULTS
from unit_testing_constants import FASTMC_CONFIG_EXAMPLE_RESULT
from unit_testing_constants import FASTMC_CONTROL_EXAMPLE_RESULT
from unit_testing_constants import FASTMC_FIELD_EXAMPLE_RESULT



def fix_parent_path():
    dir_path = os.getcwd()
    parent_dir_path = os.path.abspath(dir_path + "/../")
    if parent_dir_path not in sys.path:
        sys.path.insert(0, parent_dir_path)

fix_parent_path()

import fastmc_tools


class IsAFieldVdwLineTests(unittest.TestCase):
    def test_general_case(self):
        line = "VDW Cx        2.745   0.05948"
        result = fastmc_tools.is_a_field_vdw_line(line)
        self.assertTrue(result)

    def test_line_missing_vdw_label__should_be_False(self):
        line = "Cx        2.745   0.05948"
        result = fastmc_tools.is_a_field_vdw_line(line)
        self.assertFalse(result)

    def test_line_with_one_value__should_be_False(self):
        line = "VDW Cx        2.745"
        result = fastmc_tools.is_a_field_vdw_line(line)
        self.assertFalse(result)

    def test_line_without_label(self):
        line = "VDW        2.745   0.05948"
        result = fastmc_tools.is_a_field_vdw_line(line)
        self.assertFalse(result)



class IsAFieldChargeWeightLineTests(unittest.TestCase):
    def test_general_case(self):
        line = "Cx           12.011     +0.6645"
        result = fastmc_tools.is_a_field_charge_weight_line(line)
        self.assertTrue(result)


class ImportFastmcGuestFileComponentsTests(unittest.TestCase):
    def setUp():
        f1 = open("unit_testing_example_guest_file",'w')
        [f1.write(x) for x in GUEST_FILE_EXAMPLE]
```

```
        f1.close()


    def test_general_case(self):
        result =
fastmc_tools.import_fastmc_guest_file_components("unit_testing_example_guest_file")
        self.assertEqual(result, GUEST_FILE_COMPONENT_RESULTS)

    def tearDown():
        os.system("rm unit_testing_example_guest_file")


class ImportFastmcGuestFileTests(unittest.TestCase):
    def setUp():
        f1 = open("unit_testing_example_guest_file",'w')
        [f1.write(x) for x in GUEST_FILE_EXAMPLE]
        f1.close()


    def test_general_case(self):
        result = fastmc_tools.import_fastmc_guest_file("unit_testing_example_guest_file")
        self.assertEqual(result, GUEST_FILE_IMPORT_RESULTS)

    def tearDown():
        os.system("rm unit_testing_example_guest_file")







class WriteFastmcConfigFileTests(unittest.TestCase):
    def setUp():
        doap_job_name = "unit_test_job"
        supercell_dimensions = [2,2,2]
        cell_vectors = [[1,2,3],[4,5,6],[7,8,9]]
        cell_coordinates = [["C",1,2,3],["H",4,5,6],["N",7,8,9],["O",2,3,4],["Zn",3,4,5]]
        imcon = "3"
        fastmc_tools.write_fastmc_config_file(doap_job_name, supercell_dimensions, cell_vectors,
cell_coordinates, imcon)

    def test_general_case(self):
        f1 = open("CONFIG",'r')
        result = f1.readlines()
        f1.close
        self.assertEqual(result, FASTMC_CONFIG_EXAMPLE_RESULT)

    def tearDown():
        os.system("rm CONFIG")



class WriteFastmcControlFile(unittes.TestCase):
    def setUp():
        system_temp = "273"
        cutoff = "70.0"
        delr = "1.0"
        ewald_precision = "1d-6"
        equilibration_time = "100000"
        simulation_length = "1000000"
        guest_pp = "0.850"

        fastmc_tools.write_fastmc_control_file(system_temp, cutoff, delr, ewald_precision,
equilibration_time, simulation_length, guest_pp)


    def test_general_case(self):
        f1 = open("CONTROL",'r')
```

```
        result = f1.readlines()
        f1.close
        self.assertEqual(result, FASTMC_CONTROL_EXAMPLE_RESULT)

    def tearDown():
        os.system("rm CONTROL")


clas WriteFastmcFieldFileTests(unittest.TestCase):

    def setUp():
        f1 = open("unit_testing_example_guest_file",'w')
        [f1.write(x) for x in GUEST_FILE_EXAMPLE]
        f1.close()

        doap_job_name = "unit_test_job"
        unitcell_coordinates = [["C",1,2,3],["H",4,5,6]]
        charge_list = [["C", +1.0],["H",-1.0]]
        molecular_types = 2
        fastmc_guest_file = "unit_testing_example_guest_file"
        vdw_dict = {"C":[1.60, 0.4],"H":[ 2.3, 0.3],"Cx":[1.02,0.05],"Ox":[1.56,0.07]}
        supercell_dimensions = [[1,2,3],[2,3,4],[4,5,6]]

        fastmc_tools.write_fastmc_field_file(doap_job_name, unitcell_coordinates, charge_list,
molecular_types, fastmc_guest_file, vdw_dict, supercell_dimensions)

    def test_general_case(self):

        f1 = open("CONTROL",'r')
        result = f1.readlines()
        f1.close()
        self.assertEqual(result, FASTMC_FIELD_EXAMPLE_RESULT)

    def tearDown():
        os.system("rm unit_testing_example_guest_file")
```

unit_tests/test_molecular_mechanics_tools.py

```
import inspect
import unittest
import os
import sys
from unit_testing_constants import VDW_TEST_LINES
from unit_testing_constants import INPUT_VDW_TEST_RESULT


def fix_parent_path():
    dir_path = os.getcwd()
    parent_dir_path = os.path.abspath(dir_path + "/../")
    if parent_dir_path not in sys.path:
        sys.path.insert(0, parent_dir_path)

fix_parent_path()

import molecular_mechanics_tools

class AdjustChargesToMatchNetChargeTests(unittest.TestCase):
    def test_general_case(self):
        charge_list = [["a",0.0001],["b",0.0002],["c",-0.0004]]
        desired_charge = 0
        adjust_by = 0.0001
        result = molecular_mechanics_tools.adjust_charges_to_match_net_charge(charge_list,
desired_charge, adjust_by)
        self.assertEqaul(result, [['a', 0.00020000000000000001], ['b', 0.00020000000000000001],
['c', -0.00040000000000000002]])

class ReorderCharges(unittest.TestCase):
    def test_general_case(self):
```

```
        order_list = [0,3,4,2,1]
        charge_list = [["a",0.0001],["b",0.0002],["c",-0.0004],["d",-0.0080],["e",0.0002]]
        result = molecular_mechanics_tools.reorder_charges(order_list,charge_list)
        self.assertEqual(result, [['a', 0.0001], ['e', 0.00020000000000000001], ['d',
-0.0080000000000000002], ['b', 0.00020000000000000001], ['c', -0.00040000000000000002]])


class ImportVdwValues(unittest.TestCase):


    def setUp(self):
        test_file = open("unit_test_example_vdw_file",'w')
        [test_file.write(x) for x in VDW_TEST_LINES]
        test_file.close()

    def test_example_file_import_test__should_pass(self):
        result = molecular_mechanics_tools.import_vdw_values("unit_test_example_vdw_file")
        self.assertEqual(result,INPUT_VDW_TEST_RESULT)

    def tearDown():
        os.system("rm unit_test_example_vdw_file")
```

unit_tests/test_repeat_tools.py

```
import inspect
import unittest
import os
import sys
from unit_testing_constants import UNIT_TESTING_EXAMPLE_REPEAT_OUTPUT
from unit_testing_constants import UNIT_TESTING_EXAMPLE_REPEAT_CHARGE_IMPORT


def fix_parent_path():
    dir_path = os.getcwd()
    parent_dir_path = os.path.abspath(dir_path + "/../")
    if parent_dir_path not in sys.path:
        sys.path.insert(0, parent_dir_path)

fix_parent_path()

import repeat_tools



class WriteRepeatParameterFileTests(unittest.TestCase):

'''
write_repeat_parameter_file(repeat_config_dictionary, repeat_charge)
'''


class ParseChargesFromRepeatOutputTests(unittest.TestCase):
    def setUp():
        f1 = open(unit_testing_example_repeat_output",'w')
        [f1.write(x) for x in UNIT_TESTING_EXAMPLE_REPEAT_OUTPUT]
        f1.close()

    def test_regular_charge_file_imports(self):
        f2 = open("unit_testing_example_repeat_output",'r')
        repeat_output_lines = f2.readlines()
        f2.close()
        result = repeat_tools.parse_charges_from_repeat_out(repeat_output_lines)
        self.assertEqaul(result, UNIT_TESTING_EXAMPLE_REPEAT_CHARGE_IMPORT)

    def tearDown():
        os.system("rm unit_testing_example_repeat_output")
```

```
class IsRepeatChargeLineTests(unittest.TestCase):

    def test_regular_case__should_pass(self):
        repeat_line = ' Charge            1  of type           30  =   0.570852665223576     \n'
        result = repeat_tools.is_repeat_charge_line(repeat_line)
        self.assertTrue(result)

    def test_case_charge_is_integer__should_pass(self):
        repeat_line = ' Charge            1  of type           30  =   1     \n'
        result = repeat_tools.is_repeat_charge_line(repeat_line)
        self.assertTrue(result)
```

unit_tests/test_xyz_tools.py

```
import inspect
import unittest
import os
import sys
from unit_testing_constants import INPUT_TEST_LINE
from unit_testing_constants import INPUT_TEST_RESULT
from unit_testing_constants import SUPER_CELL_TEST_RESULT


def fix_parent_path():
    dir_path = os.getcwd()
    parent_dir_path = os.path.abspath(dir_path + "/../")
    if parent_dir_path not in sys.path:
        sys.path.insert(0, parent_dir_path)

fix_parent_path()

#Can't import until the path is fixed above
import xyz_tools


'''
self.assertRaises(ValueError)
self.assertRaises(TypeError, random.shuffle, (1,2,3))
self.assertEqual(self.seq, range(10))
self.assertTrue(x in list_w_x)

assertEqual(a, b)
assertNotEqual(a, b)
assertTrue(x)
assertFalse(x)

assertRaises(exc, fun, *args, **kwds)

assertAlmostEqual(a, b)
assertNotAlmostEqual(a, b)

'''




class GetAtomSymbolsFromCoordinateListTests(unittest.TestCase):

    def test_coordinate_has_more_than_four_items__should_fail(self):  #Detects coupling of
validation and processing.
        coordinates_with_five_items = [["A", 1.0, 2.0, 3.0, 4.0],["B", 1.0, 2.0, 3.0, 4.0]]
        self.assertRaises(ValueError, xyz_tools.get_atom_symbols_from_coordinate_list,
coordinates_with_five_items)
```

```
class GetLocationsFromCoordinateListTests(unittest.TestCase):

    def test_function_handles_one_line_list__should_pass(self):
        one_location = [["Cx",1.0,2.0,3.0]]
        result = xyz_tools.get_locations_from_coordinate_list(one_location)
        self.assertEqual(result, [[1.0,2.0,3.0]])

class FindProperAtomLabelsTests(unittest.TestCase):

    def test_function_handles_one_atom_list__should_pass(self):
        atom_list = ["Ck"]
        resolved_atom = xyz_tools.find_proper_atom_labels(atom_list)
        self.assertEqual(resolved_atom, ['C'])

    def test_returns_none_object_on_no_match(self):
        atom_list = ["Qp"]
        resolved_atom = xyz_tools.find_proper_atom_labels(atom_list)
        self.assertEqual(resolved_atom, [None])


class ReassignProperAtomLabelsTests(unittest.TestCase):

    def test_one_line_atom_coordinate__should_pass(self):
        one_atom_coordinate_line = [["Cx",1,2,3]]
        result = xyz_tools.reassign_proper_atom_labels(one_atom_coordinate_line)
        self.assertEqual(result, [('C', 1, 2, 3)])

class GetFourColumnedLinesTests(unittest.TestCase):

    def test_one_line_four_columned_line__should_pass(self):
        one_four_columned_line = ["cool and the gang."]
        result = xyz_tools.get_four_columned_lines(one_four_columned_line)
        self.assertEqual(result,[['cool', 'and', 'the', 'gang.']])

    def test_no_four_columned_lines__should_return_nothing(self):
        line_list_with_no_four_columned_lines = ["a b c d e", "one two three", "apples, carrots,
peas, corn, zebras"]
        result = xyz_tools.get_four_columned_lines(line_list_with_no_four_columned_lines)
        self.assertEqual(result,[])

class ParseUnitCellFileTests(unittest.TestCase):
    def setUp(self):
        test_file = open("unit_test_example_input_file",'w')
        test_file.write(INPUT_TEST_LINE)
        test_file.close()

    def test_example_file_import_test__should_pass(self):
        result = xyz_tools.parse_unit_cell_file("unit_test_example_input_file")
        self.assertEqual(result,INPUT_TEST_RESULT)

    def tearDown():
        os.system("rm unit_test_example_input_file")


class CreateAtomLabelSetTests(unittest.TestCase):
    def test_one_line_atom_set__should_pass(self):
        one_atom_line = [['Zn', 0.0, 4.2480000000000002, 8.4949999999999992]]
        result = xyz_tools.create_atom_label_set(one_atom_line)
        self.assertEqual(result, ["Zn"])

    def test_zero_line_atom_set__should_pass(self):
        zero_atom_line = []
        result = xyz_tools.create_atom_label_set(zero_atom_line)
        self.assertEqual(result, [])


class ReorderHAtomsToLastTests(unittest.TestCase):

    def test_one_atom_line(self):
        atoms = ["H"]
```

```
        result = xyz_tools.reorder_H_atoms_to_last(atoms)
        self.assertEqual(result,['H'])

    def test_atom_line_without_hydrogens(self):
        atoms = ["Jk","C","Lw","Hf"]
        result = xyz_tools.reorder_H_atoms_to_last(atoms)
        self.assertEqual(result,['Jk', 'C', 'Lw', 'Hf'])

    def test_atom_line_with_hydrogens(self):
        atoms = ["Jk","H","Lw","Hf"]
        result = xyz_tools.reorder_H_atoms_to_last(atoms)
        self.assertEqual(result,['Jk', 'Lw', 'Hf', 'H'])


class GetAtomQuantitiesTests(unittest.TestCase):
    def test_one_atom_line(self):
        one_atom_list = ["C"]
        result = xyz_tools.get_atom_quantities(INPUT_TEST_RESULT[0],one_atom_list)
        self.assertEqual(result, {'C': 7})

    def test_multiple_atom_line(self):
        multiple_atom_list = ["C", "Zn"]
        result = xyz_tools.get_atom_quantities(INPUT_TEST_RESULT[0],multiple_atom_list)
        self.assertEqual(result, {'C': 7, 'Zn': 2})

    def test_empty_atom_line(self):
        empty_atom_list = []
        result = xyz_tools.get_atom_quantities(INPUT_TEST_RESULT[0],empty_atom_list)
        self.assertEqual(result, {})

    def test_non_present_atom_line(self):
        non_present_atom_list = ["Xx"]
        result = xyz_tools.get_atom_quantities(INPUT_TEST_RESULT[0],non_present_atom_list)
        self.assertEqual(result,{'Xx': 0})


class IndexAndSortCoordinateListTests(unittest.TestCase):

    def test_one_line_atom_coordinate_w_multi_line_atom_list__should_pass(self):
        atom_coordinates = [["C",1,2,3]]
        atom_labels = ["C","H"]
        result = xyz_tools.index_and_sort_coordinate_list(atom_coordinates,atom_labels)
        self.assertEqual(result,([['C', 1, 2, 3]], [0]))

    def test_multi_line_atom_coordinate_w_one_line_atom_list__should_pass(self):
        atom_coordinates = [["C",1,2,3],["C",2,3,4],["H",5,6,7],["C",2,4,5]]
        atom_labels = ["H"]
        result = xyz_tools.index_and_sort_coordinate_list(atom_coordinates,atom_labels)
        self.assertEqual(result,([['H', 5, 6, 7]], [2]))

    def test_multi_line_atom_coordinate_w_multi_line_atom_list__should_pass(self):
        atom_coordinates = [["C",1,2,3],["C",2,3,4],["H",5,6,7],["C",2,4,5]]
        atom_labels = ["C","H"]
        result = xyz_tools.index_and_sort_coordinate_list(atom_coordinates,atom_labels)
        self.assertEqual(result,([['C', 1, 2, 3], ['C', 2, 3, 4], ['C', 2, 4, 5], ['H', 5, 6,
7]], [0, 1, 3, 2]))

class HydrogensPresent(unittest.TestCase):
    def test_list_with_hydrogen_present__should_return_True(self):
        atom_line_with_hydrogen = ["H","C","R","F","D"]
        result = xyz_tools.hydrogens_present(atom_line_with_hydrogens)
        self.assertTrue(result)

    def test_list_without_hydrogen_present__should_return_False(self):
        atom_line_without_hydrogen = ["C","R","F","D"]
        result = xyz_tools.hydrogens_present(atom_line_without_hydrogens)
        self.assertFalse(result)

    def test_list_with_hydrogen__like_atom_present(self):
        atom_line_with_hydrogen_like_atom = ["He","C","R","F","D"]
```

```
        result = xyz_tools.hydrogens_present(atom_line_with_hydrogen_like_atom)
        self.assertFalse(result)


class WriteAtomOrderToFileTests(unittest.TestCase):

    def setUp():
        xyz_tools.write_atom_order_to_file([3,4,5,6,8,2,3],"unit_testing_example_order_file")

    def test_example_file_creation_test__should_pass(self):
        f1 = open("unit_testing_example_order_file","r")
        result = f1.readlines()
        f1.close()
        self.assertEqual(result,['3\n', '4\n', '5\n', '6\n', '8\n', '2\n', '3\n'])

    def tearDown():
        os.system("rm unit_testing_example_order_file")


class ImportOrderListTests(unittest.TestCase):

    def setUp():
        xyz_tools.write_atom_order_to_file([3,4,5,6,8,2,3],"unit_testing_example_order_file")

    def test_example_file_creation_test__should_pass(self):
        result = xyz_tools.import_order_list("unit_testing_example_order_file")
        self.assertEqual(result,[3, 4, 5, 6, 8, 2, 3])

    def tearDown():
        os.system("rm unit_testing_example_order_file")


class ReorderCoordinatesTests(unittest.TestCase):
    def test_reorder_one_coordinate__should_pass(self):
        coordinates =[["C",1,2,3]]
        order_list = [0]
        result = xyz_tools.reorder_coordinates(order_list, coordinates)
        self.assertEqual(result,[['C', 1, 2, 3]])

    def test_reorder_multiple_coordinates__should_pass(self):
        coordinates =[["C",1,2,3],["H",2,3,4]]
        order_list = [1,0]
        result = xyz_tools.reorder_coordinates(order_list, coordinates)
        self.assertEqual(result,[['H', 2, 3, 4], ['C', 1, 2, 3]])


class IsACoordinateLineTests(unittest.TestCase):
    def test_coordinate_with_two_floats__should_be_False(self):
        coordinate_line = "C 2 3"
        result = xyz_tools.is_a_coordinate_line(coordinate_line)
        self.assertFalse(result)

    def test_coordinate_with_four_floats__should_be_False(self):
        coordinate_line = "C 2 3 4 5"
        result = xyz_tools.is_a_coordinate_line(coordinate_line)
        self.assertFalse(result)

    def test_coordinate_with_no_label__should_be_False(self):
        coordinate_line = "1 2 3"
        result = xyz_tools.is_a_coordinate_line(coordinate_line)
        self.assertFalse(result)

    def test_coordinate_with_three_floats__should_be_True(self):
        coordinate_line = "C 1.0 2.0 3.0"
        result = xyz_tools.is_a_coordinate_line(coordinate_line)
        self.assertTrue(result)
```

```
class CreateSupercellTests(unittest.TestCase):
    def test_generate_basic_supercell__should_pass(self):
        unit_cell_coordinates = [["C",0.5,0,0],["H",0,0.5,0],["N",0,0,0.5]]
        unit_cell_vectors = [[1.0,0,0],[0,1.0,0],[0,0,1.0]]
        supercell_dimensions = [2,2,2]
        result = xyz_tools.create_supercell(unit_cell_coordinates, unit_cell_vectors,
supercell_dimensions)
        self.assertEqual(result, SUPER_CELL_TEST_RESULT)
```

unit_tests/unit_testing_constants.py

```
INPUT_TEST_LINE ='''CV 16.991   00.000   00.000
CV 00.000   16.991   00.000
CV 00.000   00.000   16.991

Zn          0.0000000            4.2480000            8.4950000
C           1.4530000           10.0910000            6.9000000
Zn          4.2480000            0.0000000            8.4950000
C           0.1340000            6.4060000           10.5850000
H           2.0780000            6.0980000           13.1950000
C           9.9480000            1.5950000           15.3960000
N          15.4570000            9.0340000           13.8850000
H          14.5940000            6.4180000           12.2910000
H          15.2260000            6.3270000            7.6030000
H           2.3970000            4.7000000            6.4180000
H           2.0780000           13.1950000            6.0980000
H          14.5940000           10.5730000            4.7000000
C          11.6810000            1.7140000           10.7300000
C           5.3100000           15.2770000           10.7300000
N           0.5380000           11.6010000           10.0290000
C           3.1860000            6.7820000           14.7570000
C           2.0890000            8.3610000            2.0890000
H           2.0860000           10.0960000            6.1780000
H          12.2910000           14.5940000            6.4180000
H           1.2620000           16.2890000            9.8510000
H          14.9130000            6.0980000            3.7960000
H          14.6730000            6.4090000           15.3900000
'''
INPUT_TEST_RESULT= ([['Zn', 0.0, 4.2480000000000002, 8.4949999999999992], ['C',
1.4530000000000001, 10.090999999999999, 6.9000000000000004], ['Zn',
4.2480000000000002, 0.0, 8.4949999999999992], ['C', 0.13400000000000001, 6.4059999999999997,
10.585000000000001], ['H', 2.0779999999999998, 6.0979999999999999,
13.195], ['C', 9.9480000000000004, 1.595, 15.396000000000001], ['N', 15.457000000000001,
9.0340000000000007, 13.885], ['H', 14.593999999999999,
6.4180000000000001, 12.291], ['H', 15.226000000000001, 6.327, 7.6029999999999998], ['H',
2.3969999999999998, 4.7000000000000002, 6.4180000000000001], ['H',
2.0779999999999998, 13.195, 6.0979999999999999], ['H', 14.593999999999999, 10.573,
4.7000000000000002], ['C', 11.680999999999999, 1.714, 10.73], ['C',
5.3099999999999996, 15.276999999999999, 10.73], ['N', 0.53800000000000003, 11.601000000000001,
10.029], ['C', 3.1859999999999999, 6.782, 14.757], ['C', 2.089,
8.3610000000000007, 2.089], ['H', 2.0859999999999999, 10.096, 6.1779999999999999], ['H', 12.291,
14.593999999999999, 6.4180000000000001], ['H', 1.262,
16.289000000000001, 9.8510000000000009], ['H', 14.913, 6.0979999999999999, 3.7959999999999998],
['H', 14.673, 6.4089999999999998, 15.390000000000001]], [['CV',
16.991, 0.0, 0.0], ['CV', 0.0, 16.991, 0.0], ['CV', 0.0, 0.0, 16.991]]])


SUPER_CELL_TEST_RESULT = ([['C', 0.5, 0.0, 0.0], ['H', 0.0, 0.5, 0.0], ['N', 0.0, 0.0, 0.5],
['C', 0.5, 0.0, 1.0], ['H', 0.0, 0.5, 1.0], ['N', 0.0, 0.0, 1.5],
['C', 0.5, 1.0, 0.0], ['H', 0.0, 1.5, 0.0], ['N', 0.0, 1.0, 0.5], ['C', 0.5, 1.0, 1.0], ['H',
0.0, 1.5, 1.0], ['N', 0.0, 1.0, 1.5], ['C', 1.5, 0.0, 0.0], ['H',
1.0, 0.5, 0.0], ['N', 1.0, 0.0, 0.5], ['C', 1.5, 0.0, 1.0], ['H', 1.0, 0.5, 1.0], ['N', 1.0,
0.0, 1.5], ['C', 1.5, 1.0, 0.0], ['H', 1.0, 1.5, 0.0], ['N', 1.0,
1.0, 0.5], ['C', 1.5, 1.0, 1.0], ['H', 1.0, 1.5, 1.0], ['N', 1.0, 1.0, 1.5]], [[2.0, 0, 0], [0,
2.0, 0], [0, 0, 2.0]])
```

```
UNIT_TESTING_EXAMPLE_REPEAT_OUTPUT = [' Charge            1  of type           30  =
0.570852665223576     \n',
' Charge            2  of type           30  =   0.589745045719203     \n',
' Charge            3  of type           30  =   0.596968346503828     \n',
' Charge            4  of type           30  =   0.587149204424635     \n',
' Charge            5  of type           30  =   0.583849743410194     \n',
' Charge            6  of type           30  =   0.615430505897440     \n',
' Charge            7  of type           30  =   0.587810583565587     \n',
' Charge            8  of type           30  =   0.586744598593247     \n',
' \n',
' ESP Coulomb average:  -1.306290939756259E-016\n',
' ESP QM average:  -3.162709907394331E-014\n',
' Error of the potential:   0.315004044053435     \n']

UNIT_TESTING_EXAMPLE_REPEAT_CHARGE_IMPORT = [['Zn', 0.57089999999999996], ['Zn', 0.5897], ['Zn',
0.59699999999999998], ['Zn', 0.58709999999999996], ['Zn',
0.58379999999999999], ['Zn', 0.61539999999999995], ['Zn', 0.58779999999999999], ['Zn', 0.5867],
['Zn', 0.56769999999999998], ['Zn', 0.60060000000000002], ['Zn',
0.56779999999999997], ['Zn', 0.57050000000000001], ['C', -0.59379999999999999], ['C',
0.40500000000000003], ['C', -0.57130000000000003], ['C',
-0.21249999999999999], ['C', -0.20619999999999999], ['C', -0.24229999999999999], ['C',
0.40060000000000001], ['C', 0.4007], ['C', -0.23449999999999999], ['C',
0.39360000000000001], ['C', 0.40660000000000002], ['C', -0.21479999999999999], ['C',
-0.23419999999999999], ['C', -0.56240000000000001], ['C', -0.5776], ['C',
0.40429999999999999], ['C', -0.2319], ['C', -0.23269999999999999], ['C', -0.55349999999999999],
['C', 0.39150000000000001], ['C', -0.20710000000000001], ['C',
-0.58679999999999999], ['C', -0.23180000000000001], ['C', -0.56799999999999995], ['C',
-0.57679999999999998], ['C', 0.17760000000000001]]

EXAMPLE_JOB_OPTION_LIST = ['#----------------------\n',
'#        MAIN\n',
'#----------------------\n',
'\n',
'#Package for charge determination | CPMD_REPEAT  (VASP)\n',
'&MAIN_CHARGE_PACKAGE CPMD\n',
'\n',
'#Point charge resolutions package |  REPEAT\n',
'&MAIN_POINT_CHARGE_PACKAGE REPEAT\n',
'\n',
'#GCMC Package to use | FASTMC\n',
'&MAIN_GCMC_PACKAGE FASTMC\n', '\n', '\n', '#----------------------\n', '#        GENERAL\n',
'#----------------------\n', '\n', '#Filename for VDW params | e.g.: P\n', '&GENERAL_VDW_FILE
vanderWaal_parameters\n', '\n', '#System charge\n', '&GENERAL_SYSTEM_CHARGE 0\n', '\n',
'#Temperature in Kelvin\n',
'&GENERAL_SYSTEM_TEMP 273\n', '\n', '#----------------------\n', '#        CPMD\n',
'#----------------------\n', '\n', '#Number of Processors to use in CPMD | e.g.: 8\n',
'&CPMD_NUM_PROCESSORS 32\n', '\n', '#Orbital Convergence criteria in CPMD | e.g.: 1.0d-5\n',
'&CPMD_CONVERGENCE_CRIT_ORBITALS 1.0d-5\n', '\n',
'#Maximum number of optimization steps | e.g.: 10000\n', '&CPMD_MAX_ITERATIONS 10000\n', '\n',
'#Absolute path for pseudopotential files | e.g.:
/home/program/CHEMISTRY/CPMD/Pseudo_potentials/\n', '&CPMD_PSP_PATH
/home/program/CHEMISTRY/CPMD/Pseudo_potentials/\n', '\n', '#Default functional type | e.g.:
PBE\n', '&CPMD_FUNCTIONAL PBE\n', '\n', '#Default pseudopotential type | e.g.: SG\n',
'&CPMD_PSP_TYPE SG\n', '\n', '#Default highest orbital consideration | e.g.: P\n',
'&CPMD_LMAX_FLAG P\n', '\n', '#Cutoff for CPMD calculation in angstroms | e.g.: 70.0\n',
'&CPMD_CUTOFF 70.0\n', '\n', '#Flag for whether or not to optimize Hydrogens | True or False\n',
'&CPMD_OPTIMIZE_HYDROGENS False\n', '\n', '\n', '#----------------------\n', '#
REPEAT\n',
'#----------------------\n', '\n', '#Fit molecular(0) or periodic(1:default) system? | e.g.:
0\n', '&REPEAT_PERIODICITY  1\n', '\n', '#van der Waals scaling factor (default = 1.0) | e.g.:
1.0\n', '&REPEAT_VDW_FACTOR 1.0\n', '\n', '#Apply RESP penalties?, no(0:default), yes(1) | e.g.:
0\n', '&REPEAT_RESP_PENALTIES 0\n', '\n', '#Read cutoff radius? no(0), yes(1:default) | e.g.:
1\n', '&REPEAT_READ_CUTOFF 1\n', '\n', '#If flag above=1 provide R_cutoff next (in Bohrs) |
e.g.: 20.0\n', '&REPEAT_R_CUTOFF 20.0 \n', '\n', '#Apply symmetry restrain? no(0:default),
yes(1) | e.g.: 0\n', '&REPEAT_APPLY_SYMMETRY_RESTRAIN 0\n', '\n',
'#Use Goddard-type restrains? no(0:default), yes(1) | e.g.: 0\n', '&REPEAT_USE_GODDARD 0\n',
'\n', '#If flag above=1 then provide weight next | e.g.: 0.0\n',
'&REPEAT_GODDARD_WEIGHT 0.5\n', '\n', '\n', '\n', '#----------------------\n', '#        GCMC
\n', '#----------------------\n', '\n', '#Multiplicity of supercell in A direction | e.g.: 2\n',
```

```
'&GCMC_SUPERCELL_DIM_A 2\n', '\n', '#Multiplicity of supercell in B direction | e.g.: 2\n',
'&GCMC_SUPERCELL_DIM_B 2\n',
'\n', '#Multiplicity of supercell in C direction | e.g.: 2\n', '&GCMC_SUPERCELL_DIM_C 2\n',
'\n', '#Filename for guest information | e.g.: GUEST\n',
'&GCMC_GUEST_FILE GUEST\n', '\n', '#Required forces cutoff (Angstroms)\n', '&GCMC_CUTOFF  12\n',
'\n', '#Verlet neighbour list shell width (Angstroms)\n',
'&GCMC_DELR  1.0 \n', '\n', '#Ewald sum for electrostatics, with automatic parameter
optimisation (0 < f < 1.E . 4)\n', '&GCMC_EWALD_PRECISION 1d-6\n', '\n',
'#Number of Molecular types | e.g.: 2\n', '&GCMC_MOL_TYPES 2\n', '\n', '#See DL_POLY manual |
e.g.: 3\n', '&GCMC_IMCON 3\n', '\n', '#----------------------\n',
'#       FASTMC\n', '#----------------------\n', '\n', '\n', '#Number of Guests in fast GCMC\n',
'&FASTMC_NUM_GUESTS 1\n', '\n', '#Partial pressure of Guest\n',
'&FASTMC_GUEST_PP   0.850\n', '\n', '#fastGCMC simulation length\n', '# Simulation length and
equilibration steps that count towards average. (over all nodes)\n', '&FASTMC_SIM_LENGTH
1000000\n', '\n', '#fastGCMC equilibration time [First do this number of steps (for each
node)]\n', '&FASTMC_EQ_TIME 100000\n',
'\n', '\n']

EXAMPLE_DOAP_OPTIONS_CONFIG_DICT_RESULT = {'cpmd_optimize_hydrogens': 'False', 'gcmc_delr':
'1.0', 'general_vdw_file': 'vanderWaal_parameters',
'repeat_r_cutoff': '20.0', 'cpmd_psp_path': '/home/program/CHEMISTRY/CPMD/Pseudo_potentials/',
'repeat_read_cutoff': '1', 'gcmc_supercell_dim_a': '2',
'main_charge_package': 'CPMD', 'cpmd_num_processors': '32', 'gcmc_mol_types': '2',
'gcmc_supercell_dim_c': '2', 'general_system_charge': '0',
'gcmc_ewald_precision': '1d-6', 'repeat_resp_penalties': '0', 'gcmc_guest_file': 'GUEST',
'fastmc_num_guests': '1', 'fastmc_guest_pp': '0.850',
'gcmc_supercell_dim_b': '2', 'repeat_periodicity': '1', 'cpmd_functional': 'PBE',
'repeat_use_goddard': '0', 'repeat_vdw_factor': '1.0', 'gcmc_imcon': '3',
'main_gcmc_package': 'FASTMC', 'cpmd_cutoff': '70.0', 'cpmd_lmax_flag': 'P',
'main_point_charge_package': 'REPEAT', 'fastmc_eq_time': '100000',
'repeat_goddard_weight': '0.5', 'cpmd_convergence_crit_orbitals': '1.0d-5', 'gcmc_cutoff': '12',
'general_system_temp': '273', 'fastmc_sim_length': '1000000',
'cpmd_max_iterations': '10000', 'repeat_apply_symmetry_restrain': '0', 'cpmd_psp_type': 'SG'}


VDW_TEST_LINES = ['H         2.5711  0.0440\n', 'He        2.1043  0.0560\n', 'Li        2.1836
0.0250\n', 'Be        2.4455  0.0850\n', 'B         3.6375  0.1800\n', 'C         3.4309
0.1050\n', 'N         3.2607  0.0690\n', 'O         3.1181  0.0600\n', 'F         2.9970
0.0500\n', 'Ne        2.8892  0.0420\n',
'Pa        3.0504  0.0220\n', 'U         3.0246  0.0220\n', 'Np        3.0504  0.0190\n', 'Pu
3.0504  0.0160\n', 'Am        3.0121  0.0140\n', 'Cm        2.9631  0.0130\n', 'Bk        2.9747
0.0130\n', 'Cf        2.9515  0.0130\n', 'Es        2.9391  0.0120\n', 'Fm        2.9275
0.0120\n', 'Md        2.9168  0.0110\n', 'No        2.8936  0.0110\n', 'Lr        2.8829
0.0110\n', 'Cx        2.745   0.05948\n', 'Ox         3.017   0.17023\n']


INPUT_VDW_TEST_RESULT = {'Pu': [3.0503999999999998, 0.016], 'Ne': [2.8892000000000002,
0.042000000000000003], 'Li': [2.1836000000000002, 0.025000000000000001],
'Pa': [3.0503999999999998, 0.021999999999999999], 'Lr': [2.8828999999999998,
0.010999999999999999], 'Np': [3.0503999999999998, 0.019], 'Be': [2.4455, 0.085000000000000006],
'No': [2.8936000000000002, 0.010999999999999999], 'Bk': [2.9746999999999999,
0.012999999999999999], 'Fm': [2.9275000000000002, 0.012],
'He': [2.1042999999999998, 0.056000000000000001], 'Md': [2.9167999999999998,
0.010999999999999999], 'C': [3.4308999999999998, 0.105], 'B': [3.6375000000000002,
0.17999999999999999], 'F': [2.9969999999999999, 0.050000000000000003], 'H': [2.5710999999999999,
0.043999999999999997], 'O': [3.1181000000000001, 0.059999999999999998], 'N':
[3.2606999999999999, 0.069000000000000006], 'U': [3.0246, 0.021999999999999999], 'Ox':
[3.0169999999999999, 0.17022999999999999],
'Cm': [2.9630999999999998, 0.012999999999999999], 'Am': [3.0121000000000002, 0.014], 'Cf':
[2.9514999999999998, 0.012999999999999999], 'Cx': [2.7450000000000001, 0.059479999999999998],
'Es': [2.9390999999999998, 0.012]}


GUEST_FILE_EXAMPLE = ['Carbon dioxide\n', 'NUMMOLS 1\n', 'ATOMS  3\n', 'Cx          12.011
+0.6645\n', 'Ox          15.999     -0.33225\n',
'Ox          15.999     -0.33225\n', 'rigid  1\n', '    3    1    2    3\n', 'finish\n', 'Cx
-1.109708933         2.108201342         5.173087695\n',
'Ox   -2.119091025         1.530624172         5.162069333\n', 'Ox   -0.1003268405
2.685778512         5.184106058\n', '\n',
'VDW Cx        2.745   0.05948\n', 'VDW Ox        3.017   0.17023\n', '\n']
```

```
GUEST_FILE_COMPONENT_RESULTS = ([['Cx', '-1.109708933', '2.108201342', '5.173087695'], ['Ox', '-
2.119091025', '1.530624172', '5.162069333'], ['Ox',
'-0.1003268405', '2.685778512', '5.184106058']], [['Cx', '12.011', '+0.6645'], ['Ox', '15.999',
'-0.33225'], ['Ox', '15.999', '-0.33225']], [['Cx',
'2.745', '0.05948'], ['Ox', '3.017', '0.17023']])


GUEST_FILE_IMPORT_RESULTS = ([['Cx', '12.011', '+0.6645', '-1.109708933', '2.108201342',
'5.173087695'], ['Ox', '15.999', '-0.33225', '-2.119091025', '1.530624172', '5.162069333'],
['Ox', '15.999', '-0.33225', '-0.1003268405', '2.685778512', '5.184106058']], 'Carbon
dioxide\n',
[['Cx', '2.745', '0.05948'], ['Ox', '3.017', '0.17023']])


FASTMC_CONFIG_EXAMPLE_RESULT = ['unit_test_job 2X2X2Supercell\n',
'         0         3         5\n',
'      1.000000000000       2.000000000000       3.000000000000\n',
'      4.000000000000       5.000000000000       6.000000000000\n',
'      7.000000000000       8.000000000000       9.000000000000\n',
'C                0\n',
'      1.000000000000       2.000000000000       3.000000000000\n',
'H                0\n',
'      4.000000000000       5.000000000000       6.000000000000\n',
'N                0\n',
'      7.000000000000       8.000000000000       9.000000000000\n',
'O                0\n',
'      2.000000000000       3.000000000000       4.000000000000\n',
'Zn               0\n',
'      3.000000000000       4.000000000000       5.000000000000\n']


FASTMC_CONTROL_EXAMPLE_RESULT = ['GCMC\n', '\n', '# The state point\n', 'temperature 273\n',
'\n', '&guest 1\n', ' pressure (bar) 0.850\n', '&end\n',
'\n', '# specify cutoffs\n', 'cutoff 70.0\n', 'delr 1.0\n', '# dealing with coulombic forces\n',
'ewald precision 1d-6\n', '\n',
'#First do this number of steps (for each node)\n', 'equilibration 100000\n', '\n',
'# Simulation length and equilibration steps that count towards average. (over all nodes)\n',
'steps 1000000\n', 'numguests 100\n', '\n', 'finish']

FASTMC_FIELD_EXAMPLE_RESULT = ['Simulation of unit_test_job supercell\n', 'UNITS kcal\n',
'molecular types 2\n', '&guest Carbon dioxide\n', '\n',
'NUMMOLS 0\n', 'ATOMS 3\n', '      Cx          12.0110     0.66450  -1.1097089   2.1082013
5.1730877\n',
'      Ox          15.9990    -0.33225  -2.1190910   1.5306242   5.1620693\n',
'      Ox          15.9990    -0.33225  -0.1003268   2.6857785   5.1841061\n',
'finish\n', 'Frozen unit_test_job framework\n', 'NUMMOLS 17280\n',
'ATOMS 2\n', '       C      12.0107   1.0000 1 1\n', '       H       1.0079  -1.0000 1 1\n',
'finish\n', 'VDW  10\n', 'H       H       lj  0.30000 2.30000\n', 'C       H       lj  0.34641
1.95000\n',
'C       C       lj  0.40000 1.60000\n', 'Cx      H       lj  0.12247 1.66000\n', 'Cx      C
lj  0.14142 1.31000\n',
'Cx      Cx      lj  0.05000 1.02000\n', 'Ox      H       lj  0.14491 1.93000\n', 'Ox      C
lj  0.16733 1.58000\n',
'Ox      Cx      lj  0.05916 1.29000\n', 'Ox      Ox      lj  0.07000 1.56000\n', 'close']


EXAMPLE_XYZ_INPUT = ['CV 16.991   00.000   00.000\n',
'CV 00.000   16.991   00.000\n',
'CV 00.000   00.000   16.991\n',
'\n',
'Zn          0.0000000           4.2480000           8.4950000 \n',
'C           1.4530000          10.0910000           6.9000000 \n',
'Zn          4.2480000           0.0000000           8.4950000 \n',
'C           0.1340000           6.4060000          10.5850000 \n',
'H           2.0780000           6.0980000          13.1950000 \n',
'C           9.9480000           1.5950000          15.3960000 \n',
'N          15.4570000           9.0340000          13.8850000 \n',
'H          14.5940000           6.4180000          12.2910000 \n']
```

```
EXAMPLE_RESULTING_CPMD_INPUT_FILE = ['&CPMD\n',
'  OPTIMIZE WAVEFUNCTION XYZ\n',
'  CONVERGENCE ORBITALS\n',
'  1.0d-5\n',
'  CONVERGENCE GEOMETRY\n',
'  5.0d-4\n',
'  MAXITER\n',
'  10000\n',
'  PCG MINIMIZE\n',
'  ELECTROSTATIC POTENTIAL\n',
'&END\n',
'\n',
'&DFT\n',
'    FUNCTIONAL PBE\n',
'&END\n',
'&SYSTEM\n',
'  ANGSTROM\n',
' CELL VECTOR\n',
'     16.991000000000        0.000000000000        0.000000000000\n',
'      0.000000000000       16.991000000000        0.000000000000\n',
'      0.000000000000        0.000000000000       16.991000000000\n',
'  CUTOFF\n',
'70.0\n',
'  CHARGE\n',
'0\n',
'&END\n',
'\n',
'&ATOMS\n',
'*Zn_SG_PBE\n',
'LMAX=P\n',
'2\n',
'                0.0              4.248              8.495\n',
'              4.248                0.0              8.495\n',
'*C_SG_PBE\n',
'LMAX=P\n',
'3\n',
'              1.453             10.091                6.9\n',
'              0.134              6.406             10.585\n',
'              9.948              1.595             15.396\n',
'*N_SG_PBE\n',
'LMAX=P\n',
'1\n',
'             15.457              9.034             13.885\n',
'*H_SG_PBE\n',
'LMAX=P\n',
'2\n',
'              2.078              6.098             13.195\n',
'             14.594              6.418             12.291\n',
'&END\n']


EXAMPLE_CPMD_CHARGE_CONFIG = {'cpmd_optimize_hydrogens': 'False', 'gcmc_delr': '1.0',
'general_vdw_file': 'vanderWaal_parameters', 'repeat_r_cutoff': '20.0',
'cpmd_psp_path': '/home/program/CHEMISTRY/CPMD/Pseudo_potentials/', 'repeat_read_cutoff': '1',
'gcmc_supercell_dim_a': '2', 'main_charge_package': 'CPMD', 'cpmd_num_processors': '32',
'gcmc_mol_types': '2', 'gcmc_supercell_dim_c': '2', 'general_system_charge': '0',
'gcmc_ewald_precision': '1d-6', 'repeat_resp_penalties': '0', 'gcmc_guest_file': 'GUEST',
'fastmc_num_guests': '1', 'fastmc_guest_pp': '0.850', 'gcmc_supercell_dim_b': '2',
'repeat_periodicity': '1', 'cpmd_functional': 'PBE', 'repeat_use_goddard': '0',
'repeat_vdw_factor': '1.0', 'gcmc_imcon': '3', 'main_gcmc_package': 'FASTMC', 'cpmd_cutoff':
'70.0', 'cpmd_lmax_flag': 'P', 'main_point_charge_package': 'REPEAT', 'fastmc_eq_time':
'100000', 'repeat_goddard_weight': '0.5', 'cpmd_convergence_crit_orbitals': '1.0d-5',
'gcmc_cutoff': '4', 'general_system_temp': '273', 'fastmc_sim_length': '1000000',
'cpmd_max_iterations': '10000', 'repeat_apply_symmetry_restrain': '0', 'cpmd_psp_type': 'SG'}
```

```
EXAMPLE_CPMD_GEOMETRY_FILE = ['     276\n', 'GEOMETRY FILE / created by CPMD\n',
' Zn      0.000000000000      4.248000000000      8.495000000000              0.000000000000
0.000000000000      0.000000000000\n',
' Zn      4.248000000000      0.000000000000      8.495000000000              0.000000000000
0.000000000000      0.000000000000\n',
' Zn      8.495000000000      0.000000000000      4.248000000000              0.000000000000
0.000000000000      0.000000000000\n',
' Zn      8.495000000000      0.000000000000     12.743000000000              0.000000000000
0.000000000000      0.000000000000\n',
'  C      1.453000000000     10.091000000000      6.900000000000              0.000000000000
0.000000000000      0.000000000000\n',
'  C      0.134000000000      6.406000000000     10.585000000000              0.000000000000
0.000000000000      0.000000000000\n',
'  C      6.261000000000     15.277000000000     11.681000000000              0.000000000000
0.000000000000      0.000000000000\n',
'  C      1.453000000000      6.900000000000     10.091000000000              0.000000000000
0.000000000000      0.000000000000\n',
'  N     15.457000000000      7.957000000000      3.106000000000              0.000000000000
0.000000000000      0.000000000000\n',
'  N     13.885000000000      9.034000000000     15.457000000000              0.000000000000
0.000000000000      0.000000000000\n',
'  N      9.034000000000      3.106000000000      1.534000000000              0.000000000000
0.000000000000      0.000000000000\n',
'  N     11.601000000000      6.962000000000     16.453000000000              0.000000000000
0.000000000000      0.000000000000\n',
'  H      2.078000000000      6.098000000000     13.195000000000              0.000000000000
0.000000000000      0.000000000000\n',
'  H     14.594000000000      6.418000000000     12.291000000000              0.000000000000
0.000000000000      0.000000000000\n',
'  H     15.226000000000      6.327000000000      7.603000000000              0.000000000000
0.000000000000      0.000000000000\n',
'  H      2.397000000000      4.700000000000      6.418000000000              0.000000000000
0.000000000000      0.000000000000\n']

EXAMPLE_CPMD_GEOMETRY_IMPORT_RESULT = [['Zn', 0.0, 4.2480000000000002, 8.4949999999999992],
['Zn', 4.2480000000000002, 0.0, 8.4949999999999992],
['Zn', 8.4949999999999992, 0.0, 4.2480000000000002],
['Zn', 8.4949999999999992, 0.0, 12.743],
['C', 1.4530000000000001, 10.090999999999999, 6.9000000000000004],
['C', 0.13400000000000001, 6.4059999999999997, 10.585000000000001],
['C', 6.2610000000000001, 15.276999999999999, 11.680999999999999],
['C', 1.4530000000000001, 6.9000000000000004, 10.090999999999999],
['N', 15.457000000000001, 7.9569999999999999, 3.1059999999999999],
['N', 13.885, 9.0340000000000007, 15.457000000000001],
['N', 9.0340000000000007, 3.1059999999999999, 1.534],
['N', 11.601000000000001, 6.9619999999999997, 16.452999999999999],
['H', 2.0779999999999998, 6.0979999999999999, 13.195],
['H', 14.593999999999999, 6.4180000000000001, 12.291],
['H', 15.226000000000001, 6.327, 7.6029999999999998],
['H', 2.3969999999999998, 4.7000000000000002, 6.4180000000000001]]
```

Appendix B - Metric Analysis Code

Metric 1 – 01_map_best_fit_slopes.py

```
#! /usr/bin/python
import os
import sys
import numpy
filename = sys.argv[1]

#Return the slope of the line of best fit for a range of values from start_index to end_index
def map_slopes(file_prefix, avgbranch, window_size):
    outfile = open("%(FILE_PREFIX)02s_w%(WINSIZE)07i.asobfl" % {'WINSIZE':window_size,
'FILE_PREFIX':file_prefix} ,'w')
    sum_X,sum_XX,sum_Y,sum_XY = 0,0,0,0

    length_of_branch = len(avgbranch)

    ab_x = [x+1 for x in range(length_of_branch)]
    ab_xx = [pow(x,2) for x in ab_x]
    ab_y = avgbranch
    ab_xy = []
    for i in range(length_of_branch):
        ab_xy.append(ab_x[i]*ab_y[i])

    sum_X = sum(ab_x[:window_size-1])
    sum_XX = sum(ab_xx[:window_size-1])
    sum_Y = sum(ab_y[:window_size-1])
    sum_XY = sum(ab_xy[:window_size-1])

    slope = abs(-sum_X*sum_Y+(window_size)*sum_XY)/((window_size)*sum_XX-sum_X*sum_X)

    position = 0

    while position+window_size < length_of_branch-1:

        outfile.write(str(position+(window_size/2.0))+"   "+str(slope)+"\n")
        position = position + 1

        sum_X = sum_X - ab_x[position-1] + ab_x[position+window_size]
        sum_XX = sum_XX - ab_xx[position-1] + ab_xx[position+window_size]
        sum_Y = sum_Y -ab_y[position-1] + ab_y[position+window_size]
        sum_XY = sum_XY - ab_xy[position-1] + ab_xy[position+window_size]

        slope = abs(-sum_X*sum_Y+(window_size)*sum_XY)/((window_size)*sum_XX-sum_X*sum_X)



def import_avg_branch_file(filename):
    f1 = open(filename,'r')
    avg_branch = f1.readlines()
    for i in range(len(avg_branch)):
       avg_branch[i] = float(avg_branch[i].split()[1])
    return avg_branch


avg_branch = import_avg_branch_file(filename)

file_prefix = filename.split(".")[0]

for i in range(5000,1000000,5000):
    map_slopes(file_prefix, avg_branch, i)
```

Metric 2 – 02_standard_deviation.py

```
import os
import sys
import numpy


filename = sys.argv[1]


#The purpose of this function is to produce relative standard deviations amongst a set of data
in a windowed fashion
def generate_RSTDEV(avg_branch, file_prefix,  window_size, increment):
    avgBranch = avg_branch[:]
    maximum_point = len(avgBranch)
    window_start_position = 1
    curr_avg = 0
    curr_rSTDEV = 0


    avgs = []
    outfile = open("%(FP)02s_w%(WINSIZE)07i_%(increm)05i.stdev" %
{'FP':file_prefix,'WINSIZE':window_size,'increm':increment} ,'w')


    while((window_start_position+window_size)<=maximum_point):
        curr_rSTDEV =
numpy.std(avgBranch[window_start_position:window_start_position+window_size])
        outfile.write(str(window_start_position-1+(window_size/2.0))+" "+str(curr_rSTDEV)+"\n")
        window_start_position += increment


    outfile.close()


def import_avg_branch_file(filename):
    f1 = open(filename,'r')
    avg_branch = f1.readlines()
    for i in range(len(avg_branch)):
       avg_branch[i] = float(avg_branch[i].split()[1])
    return avg_branch


avg_branch = import_avg_branch_file(filename)


file_prefix = filename.split(".")[0]


for i in range(5000,1000000,5000):
    generate_RSTDEV(avg_branch, file_prefix, i, 5000)
```

Metric 3 – 03_max_domain.py

```
import os
import sys
import numpy


filename = sys.argv[1]



def determine_max_deviation(file_prefix, avg_branch, window_size, increment):
    avgBranch = avg_branch[:]
    max_point = len(avgBranch)
    window_start_position = 1
    curr_avg = 0


    avgs = []


    outfile = open("%(FILE_PREFIX)02s_w%(window_size)07i_%(increm)07i.mxdev" %
{'window_size':window_size, 'FILE_PREFIX':file_prefix,'increm':increment} ,'w')
```

```
    result = ""


    while((window_start_position + window_size) <= max_point):
        curr_avg = numpy.average(avgBranch[window_start_position:
(window_start_position+window_size)])
        avgs.append(curr_avg)
        temp_list = []
        win_avg = numpy.average(avgBranch[window_start_position:
(window_start_position+window_size)])
        for i in range(len(avgBranch[window_start_position:
(window_start_position+window_size)])):
            temp_list.append(abs(avgBranch[window_start_position+i]-win_avg))
        result = numpy.max(temp_list)
        outfile.write(str(window_start_position+(window_size/2.0))+" "+str(result)+"\n")
        window_start_position += increment


    outfile.close()



def import_avg_branch_file(filename):
    f1 = open(filename,'r')
    avg_branch = f1.readlines()
    for i in range(len(avg_branch)):
       avg_branch[i] = float(avg_branch[i].split()[1])
    return avg_branch



avg_branch = import_avg_branch_file(filename)

file_prefix = filename.split(".")[0]

for i in range(5000,1000000,5000):
    determine_max_deviation(file_prefix, avg_branch,i,5000)
```

Metric 4 – 04_max_gap.py

```
import os
import sys
import numpy

filename = sys.argv[1]



def determine_max_amplitudes(file_prefix, avg_branch, window_size, increment):
    avgBranch = avg_branch[:]
    max_point = len(avgBranch)
    window_start_position = 1
    curr_avg = 0


    avgs = []


    outfile = open("%(FILE_PREFIX)02s_w%(WINSIZE)07i_%(inc)07i.mxgap" % {'WINSIZE':window_size,
'FILE_PREFIX':file_prefix,'inc':increment} ,'w')


    result = ""


    while((window_start_position+window_size) <= max_point-1):
```

```
        result =
abs(numpy.max(avgBranch[window_start_position:window_start_position+window_size])-
numpy.min(avgBranch[window_start_position:window_start_position+window_size]))
        outfile.write(str(window_start_position+(window_size/2.0))+" "+str(result)+"\n")
        window_start_position += increment


    outfile.close()




def import_avg_branch_file(filename):
    f1 = open(filename,'r')
    avg_branch = f1.readlines()
    for i in range(len(avg_branch)):
       avg_branch[i] = float(avg_branch[i].split()[1])
    return avg_branch






avg_branch = import_avg_branch_file(filename)


file_prefix = filename.split(".")[0]


#window_size = int(sys.argv[2])


for i in range(5000, 1000000, 5000):
    determine_max_amplitudes(file_prefix, avg_branch, i, 5000)
```

Metric 5 – 04_max_gap.py

```
import os
import sys
import numpy


filename = sys.argv[1]


def import_avg_branch_file(filename):
    f1 = open(filename,'r')
    avg_branch = f1.readlines()
    for i in range(len(avg_branch)):
       avg_branch[i] = float(avg_branch[i].split()[1])
    return avg_branch



def get_maxes(orig_branch):
    avgBranch = orig_branch[:]
    max_point = len(avgBranch)
    position = 0
    new_max = 0
    maxes = []
    max_vals = []
    while position < max_point:
        if new_max - avgBranch[position] < 0:
            new_max = avgBranch[position]
            max_vals.append(avgBranch[position])
            maxes.append(position)
        position = position + 1
```

```
    scaling_factor = numpy.average(avgBranch[-100:])
    return maxes, scaling_factor, max_vals



def get_mins(orig_branch,maxes):
    mins = []
    avgBranch = orig_branch[:]
    for i in range(len(maxes)-1):
        lower = maxes[i]
        upper = maxes[i+1]
        current_section = avgBranch[lower:upper]
        curr_min = current_section[0]
        min_j = 0
        for j in range(len(current_section)):
            if current_section[j] < curr_min:
                curr_min = current_section[j]
                min_j = j
        mins.append(min_j+lower)

    return mins


def map_frequencies(val_list, scaling_factor):
    frequencies = []
    for i in range(len(val_list)-1):
        upper = val_list[i+1]
        lower = val_list[i]
        frequencies.append([lower,upper-lower+1])
        frequencies.append([upper,upper-lower+1])

    reduction_term = max([y for [x,y] in frequencies])/scaling_factor
    return [[x,y/reduction_term] for [x,y] in frequencies]



def write_frequencies_to_file(file_prefix,freqs,type):
    max_freq_file = open("%(FP)2s_%(ftype)3s.freq" % {'FP':file_prefix,'ftype':type} ,'w')
    for freq in freqs:
        max_freq_file.write(str(freq[0])+" "+str(freq[1])+"\n")

avg_branch = import_avg_branch_file(filename)
file_prefix = filename.split(".")[0]

maxes, scaling_factor, max_vals = get_maxes(avg_branch)
max_freqs = map_frequencies(maxes, scaling_factor)
write_frequencies_to_file(file_prefix,max_freqs,"max")
```

References

1. Herzog H., "What Future for Carbon Capture and Sequestration?" *Environmental Science & Technology* **2001**, *35* (7) 148-153.

2. Kieth D., "Why Capture CO_2 from the Atmosphere?" *Science* **2009**, *325*, 1654-1655.

3. Groisman P.; Knight R.; Easterling D.; Karl T.; Hegerl G.; Razuvaez V., "Trends in intense precipitation in the climate record." *Journal of Climate* **2005**, *18*, 1326-1350.

4. Broecker W., "Thermohaline circulation, the Achilles heel of our climate system: Will man-made CO2 upset the current balance?" *Science* **1997**, *278*, 1582-1588.

5. Rignot E., "Fast recession of a West Antartic glacier." *Science* **1998**, *281*, 549-551.

6. Rowley R.; Kostelnick J.; Braaten D.; Li X.; Meisel J., "Risk of rising sea level to population and land area." *EOS* **2007**, *88*, 105 – 107.

7. Cole S., "2009: Second Warmest Year on Record; End of Warmest Decade." *NASA* **2010** (web) http://www.giss.nasa.gov/research/news/20100121/

8. Pachauri R.; Reisinger A., "Climate Change 2007: Synthesis Report, Section 1.1: Observations of climate change." *IPCC AR4 SYR* **2007** (web) http://www.ipcc.ch/publications_and_data/ar4/syr/en/mains1.html

9. Pachauri R.; Reisinger A., "Climate Change 2007: Synthesis Report, Section 2.4: Attribution of climate change." *IPCC AR4 SYR* **2007** (web) http://www.ipcc.ch/publications_and_data/ar4/syr/en/mains2-4.html

10. Oelkers E.; Cole D., "Carbon Dioxide Sequestration: A solution to a Global Problem." *Elements* **2008**, *4*, 305 – 310.

11. Koneswaran G.; Nierenberg D., "Global Farm Animal Production and Global Warming: Impacting and Mitigating Climate Change." *Environmental Health Perspectives* **2008**, *116* (5), 578 – 582.

12. Schrag D., "Preparing to Capture Carbon." *Science* **2007**, *315*, 812 – 813.

13. D'Alessandro D.; McDonald T., “Toward carbon dioxide capture using nanoporous materials.” *Pure Appl. Chem.* **2011**, *83* (1), 57 – 66.

14. Millward A.; Yaghi O., “Metal-Organic Frameworks with Exceptionally High Capacity for Storage of Carbon Dioxide at Room Temperature.” *J. Am. Chem. Soc.* **2005***, 127*, 17998-17999

15. Anderson S.; Newell R., “Prospects for Carbon Capture and Storage Technologies.” *Annu. Rev. Environ. Resour.* **2004**, *29*, 109 – 142.

16. Yang Q.; Zhong C.; Chen J., “Computational Study of CO2 Storage in Metal-Organic Frameworks.” *J. Phys. Chem C* **2008**, *112*, 1562 – 1569.

17. Liu D.; Zheng C.; Yang Q.; Zhong C., “Understanding the Adsorption and Diffusion of Carbon Dioxide in Zeolitic Imidazolate Frameworks: A Molecular Simulation Study.” *J. Phys. Chem. C* **2009**, *113*, 5004 – 5009.

18. Banerjee R.; Furukawa H.; Britt D.; Knobler C.; O'Keeffe M.; Yaghi O., “Control of Pore Size and Fucntionality in Isoreticular Zeolitic Imidizolate Framworks and their Carbon Dioxide Selective Capture Properties.” *J. Am. Chem. Soc.* **2009**, *131*, 3875 – 3877.

19. Wang B.; Cote A.; Furukawa H.; O'Keeffe M; Yaghi O., “Colossal cages in zeolitic imidazolate frameworks as selective carbon dioxide reservoirs.” *Nature* **2008**, *453*, 207 – 212.

20. Yaghi O.; Li H.; Davis C.; Richardson D.; Groy T., “Synthetic Strategies, Structure Patterns, and Emerging Properties in the Chemistry of Modular Porous Solids.” *Acc. Chem. Res.* **1998**, *31*, 474 – 484.

21. Yazaydin A.; Snurr R.; Park T.; Koh K.; Liu J.; LeVan M.; Benin A.; Jakubczak P.; Lanuza M.; Galloway D.; Low J.; Willis R., “Screening of Metal-Organic Frameworks for Carbon Dioxide Capture from Flue Gas Using a Combined Experimental and Modeling Approach.” *J. Am. Chem. Soc.* **2009**, *131*, 18198 – 18199.

22. Cooper V.; Kong L.; Langreth D., “Computing Dispersion Interactions in Density Functional Theory.” *Physics Procedia 3* **2010**, 1417 - 1430.

23. Banerjee R.; Phan A.; Wang B.; Knobbler C.; Furukawa H.; O'Keeffe M.; Yaghi O., “High-Throughput Synthesis of Zeolitic Imidizolate Frameworks and Application to CO_2 Capture.” *Science* **2008**, *319*, 939 – 943.

24. Gupta A.; Chempath S.; Sanborn M.; Clark A.; Snurr R., “Object-oriented Programming Paradigms for Molecular Modeling.” *Molecular Simulation* **2003**, *29*, 29 – 46.

25. Rosseinsky M., “Recent developments in metal-organic framework chemistry: design, discovery, permanent porosity and flexibility.” *Microporous and Mesoporous Materials* **2004**, *73*, 15 – 30.

26. Leach A., “Molecular Modelling: Principles and Applications” *Prentice Hall* **2001**

27. Rapaport D.C., “The Art of Molecular Dynamics Simulation” *Cambridge University Press* **2004**

28. Koch W.; Holthausen M., “A Chemist's Guide to Density Functional Theory” *Wiley-VCH* **2000**

29. Cramer C., “Essentials of Computational Chemistry” *John Wiley and Sons* **2004**

30. Ewald P., “Die Berechnung Optischer und Elektrostatischer Gitterpotentiale” *Institut f. theor. Physik* **1920**, 253-287.

31. Car R.; Parrinello M., “Unified Approach for Molecular Dynamics and Density-Functional Theory.” *Physical Review Letters.* **1985**, *55* (22), 2471 – 2474.

32. Car R.; Parrinello M., “CPMD Manual.” *The CPMD Consortium* **2008** (web) http://cpmd.org/downloadable-files/no-authentication/manual.pdf

33. Campaña C.; Mussard B.; Woo T., “Electrostatic Potential Derived Atomic Charges for Periodic Systems Using a Modified Error Functional.” *J. Chem. Theory Comput.* **2009**, *5*, 2866-2878.

34. Hinchliffe A., “Molecular Modelling for Beginners.” *John Wiley and Sons* **2003**, 67 -71.

35. McQuarrie D., “Statistical Mechanics.” *Harper & Row* **1975,** 51.

36. Metropolis N.; Ulam S., “The Monte Carlo Method.” *Journal of the American Statistical Association* **1949**, *247* (44), 335 – 341.

37. Dijkstra E., “Notes on Structured Programming.” *Technological University Eindhoven: Netherlands* **1970**.

38. Dijkstra E., “Go To Statement Considered Harmful.” *Communications of the ACM* **1968**, *3* (11), 147–148.

39. Kay A., “The Early History of Smalltalk.” *ACM SIGPLAN Notices* **1993**, *28* (3), 69-95.

40. Church A., “A set of postulates for the foundation of logic.” *Annals of Mathematics* **1932**, *2* (33), 346 – 366.

41. Hughes J., “Why Functional Programming Matters.” *The Computer Journal* **1989**, *32*, 98 -107.

42. Thompson S., “Type Theory and Functional Programming.” *University of Kent* **1999**, *2* (33), 346 – 366.

43. Wadler P., “The essence of functional programming.” *Association of Computing Machinery* **1992**, 1 – 14.

44. Nilsson H.; Peterson J.; Hudak P., “Functional Hybrid Modeling from an Object-Oriented Perspective.” *Proc. of the 1st International Workshop on Equation-Based Object-Oriented Languages and Tools, Berlin, Germany* **2007,** 71–87.

45. Sanner MF., “Python: a programming language for software integration and development.” *J. Mol. Graph. Model.* **1999**, *17*, 57–61.

46. Hitz M.; Montazeri B., “Measuring Coupling and Cohesion in Object-Oriented Systems.” *Proc. Int’l Symp. Applied Corporate Computing, Monterrey, Mexico* **1995**

47. Lutz M., “Learning Python.” *O'Reilly Media* **2009**

48. van Rossum G.; Drake F., “Functional Programming HOWTO.” *Python Software Foundation* **2012**. (web) *http://docs.python.org/archives/python-2.7.2-docs-pdf-a4.tar.bz2*

49. Griffiths D.; Barry P., “Head First Programming: A Learner's Guide to Programming Using the Python Language.” *O'Reilly Media* **2009**

50. Hamill P., “Unit Test Frameworks.” *O'Reilly Media* **2004**

51. Beck K., “Extreme Programming: A Humanistic Discipline of Software Development.” *LNCS* **1998**, *1382*, 1 – 6.

52. McConnell S., “Code Complete: Second Edition.” *Microsoft Press* **2004**

53. Gamma E.; Helm R.; Johnson R.; Vlissides J., “Design patterns: elements of reusable object-oriented software.” *Pearson Eduction* **1995**

54. Rappe A.; Casewit C.; Colwell K.; Goddard A.; Skiff W., “UFF, a Full Periodic Table Force Field for Molecular Mechanics and Molecular Dynamics Simulations.” *J. Am. Chem. Soc.* **1992**, *114*, 10024 – 10035.

55. “Cambridge Crystallographic Data Centre.” (web) http://www.ccdc.cam.ac.uk/products/csd/request/

56. Johnston M.; Galvan I.; Villa-Freixa J., “Framework-Based Design of a New All-Purpose Molecular Simulation Application: The Adun Simulator.” *J Comput. Chem.* **2005**, *26*, 1647 – 1659.

57. Gillet V.; Newell W.; Mata P.; Myatt G.; Sike S.; Zsoldos Z.; Johnson A., “SPROUT: Recent Developments in the de Novo Design of Molecules.” *J. Chem. Inf. Comput. Sci.* **1994**, *34*, 207-217.

58. Stewart J., “Optimization of parameters for semiempirical methods I” *Method. J. Comput. Chem.* **1989**, *10* (2), 209.

59. Dewar M.; Zoebisch E.; Healy E.; Stewart J., “Development and use of quantum mechnical molecular models. 76. AM1: A new general purpose quantum mechnical moecular model.” *Journal of the American Chemical Society.* **1985**, *107* (13), 3902.

60. Frenkel D.; Smit B., “Understanding Molecular Simulation: from algorithms to applications.” *Academic Press* **2002**