Investigating Zero-Point Energy in a Water Trimer with Diffusion Monte Carlo

Will Solow, Skye Rhomberg, Lindsey Madison, and Eric Aaron*

Department of Computer Science, Colby College, Waterville, ME

E-mail: whsolo22@colby.edu

Abstract

We give an overview of the Diffusion Monte Carlo (DMC) algorithm and its applications into finding solutions to the Schrödinger Equation when no analytical solution is available. An implementation of the DMC algorithm is presented with a specific focus on understanding the behavior of a three-water molecule system. Given the same data structure, a four-dimensional NumPy array, we demonstrate how a fully vectorized implementation is at least 35 times faster than a traditional for loop implementation, and how to validate an inherently less obvious implementation. In such molecularly complicated systems, we show how the role of timestep factors into both the population of walkers and the calculated Zero-Point energy of the system. Following up, we give necessary criteria to find the Zero-Point energy to the desired precision, and show the wave function, a normal distribution of bond lengths or angles within the studied molecular system. Finally, we show the difficulty of equilibrating a complicated system like the water trimer, and we illustrate techniques for how build an equilibrated system when the DMC algorithm is not sufficient on its own when the system is initialized with random values.

Introduction

The Schrödinger equation can be solved for only the simplest of systems. Often, in Chemistry, we wish to know the result of the Schrödinger equation for more complicated molecular systems, such as a single water molecule and a water trimer system. However, due to the complexities of these systems, the Schrödinger equation is not able to be solved analytically, and so no resulting wave function can be obtained.

Instead, we turn to the Diffusion Monte Carlo (DMC) algorithm, first introduced by Anderson. We give an efficient, generalizable, and validated NumPy implementation of the DMC algorithm in order to calculate the Zero-Point energy and wave function of a single water molecule system and water trimer system. To our knowledge, this is the first time that the water trimer system has been studied using DMC methods.

We give evidence to support the importance of time step when performing calculations, both in the capacity of the Zero-Point energy and the wave function. We show how time step impacts the population of walkers in the DMC algorithm, an interesting discovery, and how a smaller timestep alleviates this problem.

We conclude with the importance of computational efficiency and give an empirical analysis of two different implementations of the DMC algorithm with specific focus on how the potential energy function is calculated. In any computationally heavy experiment, efficiency is important. We show a strategy for leveraging vectorization techniques to arrive at a meaningfully faster implementation. In addition, we give tools to validate these vectorization techniques given that the code is less inherently obvious than a more typical for loop implementation.

Related Work

This semester, we code a python based implementation of the Diffusion Monte Carlo algorithm originally presented in Anderson 1975.[?] Our implementation of the DMC algorithm

can be directly attributed to Anderson's work, as he was one of the first to propose such a method. At the time, other methods were seen as more computationally efficient for a given level of accuracy. We take Anderson's algorithm and refine it in the scope of being efficient for computing the ground level energy and bond strength of Clathrate Hydrates.

While Anderson was initially only concerned with the calculation of the ground energy of H3, his simplifying approach to the 1D and 6D systems greatly influenced our approach in initial versions of our implementation. We improve on his work by designing a more efficient, vector-based implementation using the numpy library provided by python. We hope to extend this approach to the task of modelling more complicated Clathrate Hydrates which require an increased computational load. As Anderson points out, the Schrödinger Equation is only analytically solvable for certain, small systems. His DMC algorithm provides such a way to approximate solutions to complex systems. The best solution to modelling Clathrate Hydrates, given the lack of an analytical solution, is an open question. We hope to build on Anderson's work to provide a computationally efficient method for the solution of this problem.

Since Anderson, significant work has been done nuancing DMC algorithms and addressing some of their weaknesses. In particular, systemic bias based upon time-step has been documented in naive DMC implementations.[?] Importance sampling, which involves picking random values for the diffusion of walkers from an alternative probability distribution to avoid particularly bad results, is used by various authors to mitigate time-step error as well as to increase computational efficiency.[? ?] While our implementation more closely follows Anderson's than those presented in Reynolds or Urimgar, our efforts to improve computational efficiency and expand the scope of applicability for DMC algorithms are far from alone.

Modeling

The Model

At a high level, in each loop of the simulation, the DMC algorithm first calculates the reference energy of the system (one can think of this as the average zero-point energy). Then, the algorithm takes an array of walkers, each representing a possible state of the system, and propagates the atoms in each walker within a normal distribution based on the time step. After propagation, the potential energy of each walker is calculated. Walkers with potential energy higher than the reference energy are deleted, based on a given probability, while walkers with reasonable potential energy are replicated. Over many time steps, the reference energy starts to converge to an experimentally found value, and by taking a 1000 step rolling average, one can compute the Zero-Point energy of the system.

Given the pseudo-randomness leveraged in code-based implementations of the DMC algorithm, this serves to approximate every possible configuration of the system, within the bounds in which it would normally exist in nature. The resulting positions of the atoms in the system serves as the solution to Schrödinger's wave function equation, and we see that the bond lengths and bond angles between atoms appear within a normal distribution of their equilibrium values.

Over time, the rolling average of the potential energy of all systems that are valid converges to the zero-point energy, or the ground state energy of the system. This produces an approximate solution of the Schrödinger equation, which is what the DMC algorithm is modelling.

Implementation

In our implementation of DMC, we write a script-based implementation in Python, relying heavily on the NumPy library, which provides "vectorization" for fast operations on large matrices—that is, it allows processors to perform concurrent operations while looping over

matrices, greatly reducing runtime compared to conventional looping. The most important variable in the code is a 4D array which stores the Cartesian coordinates of each atom in each molecule in each walker. As we move through the simulation loop of the algorithm, we operate on the walker array to delete and replicate walkers as required by the algorithm.

Notably, the function that calculates potential energy is drastically different from molecular system to molecular system, so the main simulation loop uses calls to the potential energy function for the purpose of extensibility. From system to system, the only other varying part of our code is how we propagate each walker. The distribution of possible propagations for various atoms depends upon their mass, and so subroutine to propogate a particular walker is contingent upon the particular order of atoms therein. This could potentially change with each system, although we hope to find a data structure that allows our code to remain efficient without needing to change how the walkers are propagated at each step. We find some promise in generating the array of propogations—i.e. how far each atom moves in each coordinate—wholesale by stacking together many copies of a list of the walker's atomic masses up to the dimensionality of our 4D array. Otherwise, the entire model is controlled by the initial variables of the simulation.

Through the simulation loop, we rely heavily on the broadcasting features of NumPy to different array dimensions to figure out which walkers are to be deleted or replicated. We like to think that this is done quite cleanly, and do it in a mere 11 lines of code after the walkers are propagated. This efficiency is appealing and has been why we are hesitant to walk away from the 4D array based implementation that we provide. The main disadvantage of this method becomes apparent when considering how to model complicated molecular structures like Clathrate Hydrates, which don't have a consistent number of atoms per molecule as they exhibit heterogeneity of molecules within a walker. This means that one of the four dimensions of our array is "ragged." There are a few ways to work this raggedness into our data structure, but they are all patches rather than ideal solutions.

Code Validation

In any implementation of an algorithm, rigorous testing is required to ensure that the code is producing accurate results. Ideally, data generated from a simulation would be corroborated with data collected experimentally. However, due to the nature of the study of particles, outside of the simplest systems, it is difficult to accurately calculate the ground state energy. As such, results generated from the DMC algorithm cannot be verified against real world data.

Instead, we turn to manual validation of the code. An outline of the process is as follows: a set of walkers is generated. The validator calculates the potential energy and the reference energy. Using a randomly generated set of thresholds, the validator determines which walkers should be deleted and replicated, and then compares the final array to the final array produced by the algorithm. To be sure of correctness, this process was repeated multiple times with sets of ten walkers over five time steps. These groups of calculations should be comprehensive enough to validate the correctness of the algorithm due to the stochastic nature of the simulation, and generalize well to larger groups of walkers.

As a final measure, the implementaion of the DMC algorithm was tested to model the carbon monoxide bond, given that the Zero-Point energy of the CO bond is well known. When tested over multiple simulations, the 1,000 step rolling average varied on the order of $1 \cdot 10^{-4}$, with a variance on the order of $1 \cdot 10^{-3}$. Based on prior DMC implementations, this amount of inaccuracy is normal.

Simulations

We start with an empirical analysis of a fully vectorized potential energy function and compare it to a for loop implementation. These simulations were done on the single water molecule system and so only the intramolecular potential energy function is considered. With a time step that yields a convergent walker population, in this case we considered

dt=.1, we run simulations on systems with different initial walker populations and simulation length to understand how the runtime of the simulation varies under these parameters. With this information we demonstrate the importance of a vectorized approach. While sufficient data is shown below, we see that across all simulations, the improvement is roughly constant.

However, given that this improvment is upward of 30 times faster than a for loop implementation, we see that it is useful to use. Large DMC simulations can take a few hours to run, especially as we consider more complicated systems with more molecules. Thus, this magnitude of improvement becomes drastically more meaningful as run time escalates. (graphs to come)

Now, we show how the time step chosen impacts the convergence of the walker population and the calculated zero point energy. We ran simulations over a variety of time steps and walker populations. To eliminate some of the "noise" in these simulations, given that the DMC algorithm is stochastic in nature, we take the average over 10 or 20 simulations. By doing this, we demonstrate necessary conditions to calculate a zero-point energy that is accurate on the order of five decimal places. We also present the resulting wave function which is represented by the normal distribution of the Oxygen-Hydrogen bond lengths about equilibrium. (graphs coming)

Finally, we perform the same simulation for the water trimer system. The water trimer system is significantly more complicated than a single water molecule system given that we enforce distances between all nine atoms in the system. As such, the conditions for convergence, and finding a reasonable value, are much stricter. Our resulting wave function is a graph of the bond angles between all oxygen atoms in the system. In a water trimer, we would expect to see an equilibrium bond angle of 60 degrees, given that the oxygen angles align themselves in an equiangular triangle.

Discussion

When creating a piece of code in the field of Computational Science, its worth is often based on how extendable it is to other projects of interest. In the case of our work, we provide a fast, script-based implementation of the DMC algorithm by leveraging the vectorization provided in the NumPy library. Currently, our work relies on the assumption that the molecules in the modeled system are homogeneous. Clearly, this is not a particularly extendable piece of code, as there are many other interesting systems made up of hetereogeneous molecules.

This begs us to consider how to create generalizable code that can be used in a variety of circumstances without sacrificing efficiency. Observe that NumPy works well when dealing with arrays of many dimensions. As soon as we change the sizes of the molecules in a walker, the dimensions of our array are no longer even, and so NumPy cannot cleanly vectorize the solution in the 4D data structure that we have presented up to this point. From a programming point of view, it would not be particularly difficult to represent each walker on a singular array dimension. However, this also reduces the extendablility of our code as it makes the potential energy function extraordinarily difficult to calculate, given that each molecule and atom corresponds to a particular index on the same array dimesion. It also makes initializing the walker array cumbersome and unintuitive, as the user would have to memorize a potentially very long string of atoms and know their indices to facilitate the potential energy calculations both within and between molecules in a particular walker.

We further plan to extend our implementation to include importance sampling which appears in ( ). This will allow us to weight different values of the reference energy. As such, we hope that this allows us to find more accurate data using less computationally expensive methods such as a smaller time step and larger population of walkers.

Conclusion