Calculating the Zero Point Energy of Ethane Using the Diffusion Quantum Monte Carlo Method

Simon Neidhart

Supervisor: Prof. Dr. Stefan Goedecker

University of Basel

March 27, 2021

Contents

Chapter 1

Introduction

Monte Carlo methods are almost as old as computers. Its first application was to simulate neutron transport in fissionable material. Here, the outcome of collisions with the material as well as the different types of collisions or reactions are the random parts of the problem and thus require random numbers to simulate in a computer. This simulation is very intuitive as the computer simulates directly what happens in nature. The mathematical reason that this works, is that the underlying integral is in fact solved by numerical integration using these random numbers. Once people realized this, the application of Monte Carlo methods started to drift away from the obvious problems such as fission and more towards any problem that needs to solve a high dimensional integral. It is very important to keep this mathematical foundation in mind as the most intuitive approaches are often not the most efficient to do a Monte Carlo calculation.
Inspired by this, we exploit the similarity between the Schrödinger equation and the diffusion equation to develop the Diffusion Quantum Monte Carlo (DQMC) algorithm. It is rather obvious where the the randomness lies in pure diffusion and a short derivation will show that this can also be applied to solving the many-body Schrödinger equation. In the field of quantum mechanics, Monte Carlo simulations can be very useful as there are a lot of high dimensional integrals that need to be solved. Of the many approaches that exist in quantum Monte Carlo, DQMC is the most versatile.
In this work, the aim is to calculate zero point energies with the developed DQMC algorithm. In the first step, the algorithm is tested using the well known quantum harmonic oscillator. Then, it is applied to the molecule of ethane to calculate its zero point energy. The result is then compared to a reference calculation and we conclude that the algorithm performed very well.

Chapter 2

Theoretical Background

This chapter covers the theory behind the developed DQMC algorithm from a mathematical and physical standpoint. The ingredients we need range from statistics and random numbers to quantum physics an the Schrödinger equation.

2.1 Random Number Generation

For all Monte Carlo methods we need lots of random numbers. As we know, computers are deterministic machines so we need to resort to pseudo random numbers. These random numbers are not truly random because they can be predicted if one knows the initial parameters of the random number generator. However, this is not a problem here because for the Monte Carlo application the random numbers do not need to be truly random, but only in the sense of their probability distribution for a very large number of random numbers. For this application a good pseudo random number generator (PRNG) is sufficient and has the advantage that it can generate lots of random numbers fast.
For the DQMC algorithm described later, we need random numbers from two different probability distributions: $\mathcal{U}([0,1])$ and $\mathcal{N}(\mu,\sigma)$. For the $\mathcal{U}([0,1])$ random numbers the we use Fortran's intrinsic function `RANDOM_NUMBER` which internally uses the xoshiro256** PRNG [1], a variant of the xorshift PRNG. For the $\mathcal{N}(\mu,\sigma)$ random number, we first generate $\mathcal{U}([0,1])$ random numbers and then transform them to $\mathcal{N}(0,1)$ random numbers using the Box-Muller algorithm (Appendix A) before we transform them to the desired mean and variance with the formula

$$x' = x\sigma + \mu. \tag{2.1}$$

2.2 Monte Carlo Methods

The core of the Monte Carlo approach is to do numerical calculations by using lots of random numbers distributed according to a desired probability distribution $p(x)$ to approximate expectation values or integrals with averages over many trials.

$$\langle \mathcal{A} \rangle = \sum_{x\in\Omega} \mathcal{A}(x)p(x) \approx \frac{1}{N}\sum_{i=1}^{N} \mathcal{A}(x_i), \tag{2.2}$$

or

$$\int_{x\in\Omega} \mathcal{A}(x)p(x)d^n x \approx \frac{1}{N}\sum_{i=1}^{N} \mathcal{A}(x_i), \tag{2.3}$$

where x_i are the random numbers generated according to the probability distribution $p(x)$. Because doing a sum or integral over the entire space Ω is often not possible because the space is too large, one generates the x_i from a Markov Chain. This means that from a given x_i, $x_{i+1} = f(x_i, i, \text{"noise"})$ can be generated using a stochastic process. To ensure that the x_i follow the probability distribution $p(x)$ a acceptance-rejection step needs to be introduced. Namely, if $\alpha = \frac{p(x_{i+1})}{p(x_i)} > 1$ the new value x_{i+1} always gets accepted and if $\alpha < 1$ the new value x_{i+1} only gets accepted with probability α. Put together, these steps leads to the metropolis algorithm (Appendix B). These ingredients form the basis of most Monte Carlo methods including the here presented DQMC algorithm.

2.3 Diffusion Quantum Monte Carlo

For the DQMC algorithm we use the connection between the Schrödinger equation

$$H\psi(\boldsymbol{x}, t) = E_T\psi(\boldsymbol{x}, t), \tag{2.4}$$

where $H = -\frac{1}{2}\nabla^2_{\boldsymbol{x}} + V(\boldsymbol{x})$ is the Hamiltonian and and $\psi(\boldsymbol{x}, t)$ the many particle wave function and the diffusion equation

$$\frac{\partial}{\partial t}\rho(\boldsymbol{x}, t) = \frac{1}{2}\nabla^2_{\boldsymbol{x}}\rho(\boldsymbol{x}, t) + (E_T - V(\boldsymbol{x}))\rho(\boldsymbol{x}, t). \tag{2.5}$$

Comparing these two equations, we notice that for $\frac{\partial}{\partial t}\rho(\boldsymbol{x}, t) = 0$ the diffusion equation satisfies the Schrödinger equation Then (2.5) satisfies the Schrödinger equation

$$-\frac{1}{2}\nabla^2_{\boldsymbol{x}}\rho(\boldsymbol{x}, t) + V(\boldsymbol{x})\rho(\boldsymbol{x}, t) = E_T\rho(\boldsymbol{x}, t). \tag{2.6}$$

This means that we can simulate diffusion until a stationary point is reached when $\frac{\partial}{\partial t}\rho(\boldsymbol{x}, t) = 0$. To simulate this diffusion we use the known short time propagator of (2.5) for moving a particle form position $\boldsymbol{x}$ to position $\boldsymbol{y}$ during a short time step Δt

$$G(\boldsymbol{x}, \boldsymbol{y}; \Delta t) = \frac{1}{\sqrt{2\pi\Delta t}} e^{-(\boldsymbol{x}-\boldsymbol{y})^2/(2\Delta t)} e^{-\Delta t(V(\boldsymbol{y})-E_T)}. \tag{2.7}$$

This propagator has two parts. The first part $\frac{1}{\sqrt{2\pi\Delta t}} e^{-(\boldsymbol{x}-\boldsymbol{y})^2/(2\Delta t)}$ is pure diffusion and corresponds to a Gaussian distribution centred around $\boldsymbol{x}$ giving a smaller value, the further $\boldsymbol{y}$ is away from $\boldsymbol{x}$. The second part $q = e^{-\Delta t(V(\boldsymbol{y})-E_T)}$ represents a branching process. If $V(\boldsymbol{y}) > E_T$ then $q < 1$ and if $V(\boldsymbol{y}) < E_T$ then $q > 1$. Therefore q can be used to decide if a move was beneficial to decreasing the energy or not, similar to the acceptance-rejection step in the Metropolis algorithm.
With these ingredients we can construct our DQMC algorithm using a population of walkers and simulating diffusion on them. The branching process can be implemented by letting walkers die, survive or reproduce according to the value of q. If $q < 1$, meaning the diffusion move

has put the walker above the barrier of E_T, we let the walker die with probability $1 - q$ or survive with probability q. This ensures that this classically forbidden move can happen with an exponentially decreasing probability, guaranteeing that we can in principle explore the entire potential energy surface $V(\boldsymbol{x})$. If on the other hand $q > 1$, meaning the diffusion move has put the walker below the barrier of E_T, we let the walker survive and give birth to $\lfloor q - 1 \rfloor$ new walkers and an additional walker with the probability $q - \lfloor q \rfloor$. This makes walkers that live in a region where $V(\boldsymbol{x})$ is small compared to E_T reproduce and walker that live in a region where $V(\boldsymbol{x})$ is large compared to E_T die out over time.
The last ingredient we need is to adjust the energy barrier E_T to control the population of walkers. There are multiple ways to do this but generally speaking we need to decrease E_T if the number of walkers is growing and increase E_T if the number of walkers is decreasing. When we arrive at a point where the number of walkers and therefore also E_T remains stable, we have found the energy eigenvalue of the Schrödinger equation.

2.4 Unguided DQMC

Putting all this theory together, we get the unguided DQMC algorithm.

Algorithm 1 Unguided DQMC

```
 1: procedure DQMC($steps, \Delta t, V(x)$)
 2:     initialize $walkers$
 3:     for $i = 1, steps$ do
 4:         for all $x$ in $walkers$ do
 5:             draw $y \sim \mathcal{N}(y, \sqrt{\Delta t})$
 6:             $x = x + y$                                  ▷ Moving the walker
 7:             $q = exp(-\Delta t(V(x) - E_T))$
 8:             $children = \lfloor q \rfloor$               ▷ $children$ includes the surviving parent
 9:             draw $r \sim \mathcal{U}([0, 1])$
10:             if $(q - \lfloor q \rfloor > r)$ then
11:                 $children = children + 1$                ▷ $children = 0$ means the walker dies
12:             end if
13:             make $children$ copies of $walker$ $x$ for the next step
14:         end for
15:         update $E_T$                                     ▷ According to (2.8) or (2.9)
16:     end for
17: end procedure
```

2.4.1 Walker Initialization

There are several ways to initialize walkers. One is to distribute a given number of them randomly in a given simulation box. This can be done for every dimension individually by drawing uniform random numbers within the specified box bounds. Alternatively, one can start with the walkers evenly spaced along a grid in every dimension or start with all walkers in the minimum of the potential energy surface if this is known.

For the calculations of the zero point energy, it makes sense to start with all the walker in the optimized geometry of the system that is also used as a reference energy to calculate the zero point energy.

2.4.2 The Updating of E_T

The update step of the energy barrier E_T (line 15) can be done in different ways. The most trivial way is just to check if the if the walker population has increased or decreased and then divide or multiply E_T by a constant factor $\alpha > 0$. This method has two distinct problems. First, if the initial guess for E_T is way to large, this method decreases it to slow and in the meantime the population of walkers grows exponentially slowing down the calculation and eventually exceeding the memory limit. On the other hand, if the inital guess is too low, the walkers die out very fast and the simulation stops. Second, if E_T gets close to its true value the multiplication or division by α changes it to much and we can never get a very accurate result. Therefore, we need a formula that adjusts E_T depending on how fast the walker population changes. One such formula [2] adjusts E_T according to

$$E_T(t) = E_T(t - A\Delta t) - \frac{\xi}{A\Delta t} \log \frac{N_w(t)}{N_w(t - A\Delta t)}, \tag{2.8}$$

where $N_w(t)$ is the number of walkers at the discrete time t and $\xi > 0$ is a damping parameter to prevent large fluctuations in E_T. This formula does not adjust E_T in every time step but only every A time steps. As reported in [2], good values to use for this formula are $\xi = 0.1 - 0.05$ and $A = 5 - 10$.
A different approach [3] uses the potential energy of the current configuration to update E_T according to

$$E_T(t) = \bar{V} - \alpha \frac{N_w(t) - N_w(0)}{N_w(0)}, \tag{2.9}$$

where $\bar{V}$ is the ensemble average of the potential over all the walkers. For the damping constant α it is recommended to use a value close to $1/\Delta t$. This appears to be quite inefficient because one needs to calculate the potential again for every walker but this is actually not the case since we can use the values already calculated (line 7) and let them contribute to the ensemble average children times. If we were to do this between line 12 and 13, it is quite efficient.

2.5 Guided DQMC

Guided DQMC introduces a trial wave function $\psi_T(\boldsymbol{x})$ to help the algorithm converge faster. This leads to two major changes in the algorithm. [4]
The update step of the walker position (line 6) is now no longer pure diffusion, but has an added term called the drift velocity $\boldsymbol{v}(\boldsymbol{x}) = \nabla_{\boldsymbol{x}}\psi_T(\boldsymbol{x})/\psi_T(\boldsymbol{x})$. The step proposing a new positions for a walker $\boldsymbol{x}_i$ is now

$$\boldsymbol{x}_{i,F} = \boldsymbol{x}_{i,I} + \boldsymbol{v}(\boldsymbol{x}_i)\Delta t + \boldsymbol{y_i}, \tag{2.10}$$

where $\boldsymbol{x}_{i,F}$ is the final and $\boldsymbol{x}_{i,I}$ is the initial position of the walker. $\boldsymbol{y}_i$ is a normal distributed random number with mean $\boldsymbol{x}_{i,I}$ and standard deviation $\sqrt{\Delta t}$. The addition of the drift velocity $\boldsymbol{v}(\boldsymbol{x}_i)$ to the diffusion term moves the walker towards a region where $|\psi_T(\boldsymbol{x}_i)|$ increases.

The birth, survival and death process of the walkers also needs to be adapted. All walkers get a weight w_i which is updated every step according to

$$w_{i,F} = w_{i,I} \exp\left(-\Delta t(E_L(\boldsymbol{x}_{i,F}) + E_L(\boldsymbol{x}_{i,I}))/2 - E_T)\right), \tag{2.11}$$

where $E_L(\boldsymbol{x}) = (H\psi_T(\boldsymbol{x}))/\psi_T(\boldsymbol{x})$ is the local energy with H the Hamiltonian. Depending on the form of the trial wave function $\psi_T(\boldsymbol{x})$, the calculation of the local energy and the drift velocity can be done analytically (e.g. for Gaussian trial wave functions) or has to be done numerically (Appendix C).
With these non-integer weights, one can implement the split-join algorithm [5] which is an improvement to the implementation of the branching process used before. In each iteration the weight for each walker is updated according to (2.11). Walkers with a weight $w_i > 2$ split into $\lfloor w_i \rfloor$ walkers with new weights $w_i/\lfloor w_i \rfloor$. For two walkers α and β with weight $w_\alpha < 1/2$ and $w_\beta < 1/2$ one keeps walker α with probability $w_\alpha/(w_\alpha + w_\beta)$ and walker β otherwise. The surviving walker gets the new weight $(w_\alpha + w_\beta)$. All walkers with weight $1/2 \leq w_i \leq 2$ survive and keep their weight. This algorithm keeps the total weight of all walkers $W^{(k)} = \sum_{i=1}^{N_w^{(k)}} w_i^{(k)}$ constant during iteration k and limits the duplication of walkers that would occur with the standard branching process.
The adjusting of the trial energy E_T from iteration k to $k+1$ can also be improved by using the formula

$$E_T^{(k+1)} = E_0^{(k)} + \xi \log(W^{(k)}/W^{(0)}), \tag{2.12}$$

where $\xi > 0$ is a damping parameter, $W^{(0)}$ is the sum of weight at the start of the simulation and

$$E_0^{(k)} = \frac{\sum_{i=1}^{N_w^{(k)}} w_i^{(k)} E_L(\boldsymbol{x}_i^{(k)})}{\sum_{i=1}^{N_w^{(k)}} w_i^{(k)}}, \tag{2.13}$$

is the current estimation of the ground state energy E_0. At the end of the simulation, this estimate can be improved by averaging it over may iterations after the equilibration phase giving the final result of the calculation.

Chapter 3

Results

3.1 The Quantum Harmonic Oscillator

To test the algorithm, we first let it run with the potential of the d-dimensional quantum harmonic oscillator

$$V(\boldsymbol{x}) = \sum_{i=1}^{d} \frac{1}{2} m \omega_i^2 x_i^2, \tag{3.1}$$

where m the mass and $\omega_i = \sqrt{\frac{k_i}{m}}$ the angular frequencies with k_i the force constants. The quantum harmonic oscillator has a known analytical solutions for the quantum mechanical ground state energy in atomic units

$$E_0 = \sum_{i=1}^{d} \frac{1}{2} \omega_i, \tag{3.2}$$

and for the ground state wave function

$$\psi_0(\boldsymbol{x}) = \prod_{i=1}^{d} \left(\frac{m\omega_i}{\pi} \right)^{1/4} e^{-\frac{1}{2} m \omega_i x_i^2}. \tag{3.3}$$

These theoretical values allow us to compare and verify the results of our simulation.
In the following we present the results of the simulation for these parameters.

$d = 12$
$\boldsymbol{k} = (2.0, 1.1, 0.4, 1.8, 3.2, 1.0, 3.4, 1.4, 1.1, 0.6, 1.9, 0.6)$
$m = 1$
Initial walker positions: $(0, 0, ..., 0)$
$\Delta t = 0.1$
$N_w^{(0)} = 10^4$
$E_T^{start} = 8$
$\xi = 0.1$ $A = 1$ (using (2.8) to update the trial energy)

With these parameters and equation (3.2), the theoretical value for the ground state energy E_0 is calculated to be 7.11475.

3.1.1 Comparison of guided and unguided DQMC

For guided DQMC the choice of the trial wave function very much determines the convergence speed. The perfect trial wave function is the actual ground state wave function as in equation (3.3). This is a product of d Gaussian with means $\boldsymbol{\mu} = \mathbf{0}$ and variances $\sigma_i^2 = 1/(m\omega_i) = 1/\sqrt{mk_i}$ for $i = 1, ..., d$. Using this perfect trial wave function, the DQMC algorithm converges in the fist iteration step and gives the exact ground state energy up to machine precision. Obviously, for more realistic problems this the exact ground state wave function is not known and therefore cannot be used as a trial wave function. There, one needs to use the best possible approximation of the actual wave function as a trial wave function. To simulate this, we used trial wave functions with randomly shifted values for $\boldsymbol{\mu}$ and $\boldsymbol{\sigma^2}$, namely

$$\tilde{\mu}_i = \mu_i + c\,\mathcal{U}([-1, 1]), \tag{3.4}$$

and

$$\tilde{\sigma}_i^2 = \sigma_i^2 + c\,\mathcal{U}([-1, 1]), \tag{3.5}$$

for $i = 1, ..., d$. The parameter c determines the quality of the trial wave function. $c = 0$ corresponds to the perfect trial wave function.

c	$\bar{E}_0$ (averaged over the last 8000 steps)
unguided	7.11159 standard deviation: $2.0666 \cdot 10^{-2}$
1	does not converge
0.1	7.12980 standard deviation: $3.7232 \cdot 10^{-3}$
0.01	7.11688 standard deviation: $3.8956 \cdot 10^{-4}$
0.001	7.11497 standard deviation: $3.6661 \cdot 10^{-5}$
0.0001	7.11475 standard deviation: $6.7776 \cdot 10^{-6}$

Table 1: Averaged ground state energy for different qualities of the trial wave function after a burn-in phase of 2000 time steps.

One can clearly see that the quality of the guiding wave function drastically improves the performance of the DQMC algorithm in terms of the accuracy but also in terms of the fluctuations. The gain in accuracy and the decrease in the standard deviation are roughly proportional to the quality of the trial wave function. However, if the guiding wave function is chosen poorly, the algorithm might not converge at all or one would get a better result using unguided DQMC.

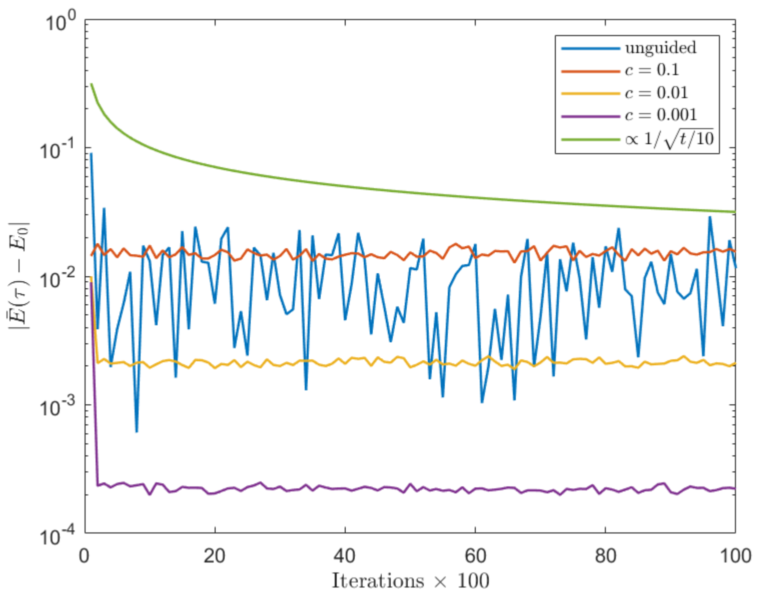

Figure 1: Convergence of the ground state energy for different qualities of the trial wave function (energies averaged over 100 iterations).

Here, we can see that the speed of the convergence does not improve much with the quality of the trial wave function. All simulations show a better convergence than the typical convergence of Monte Carlo simulations of $1/\sqrt{t}$ for t being the number of time steps calculated. The fluctuations however, differ significantly and improve with a better trial wave function. However, if we take the average of the blue curve for the unguided DQMC algorithm, we actually get a better result than we would get for guided DQMC with $c = 0.1$.

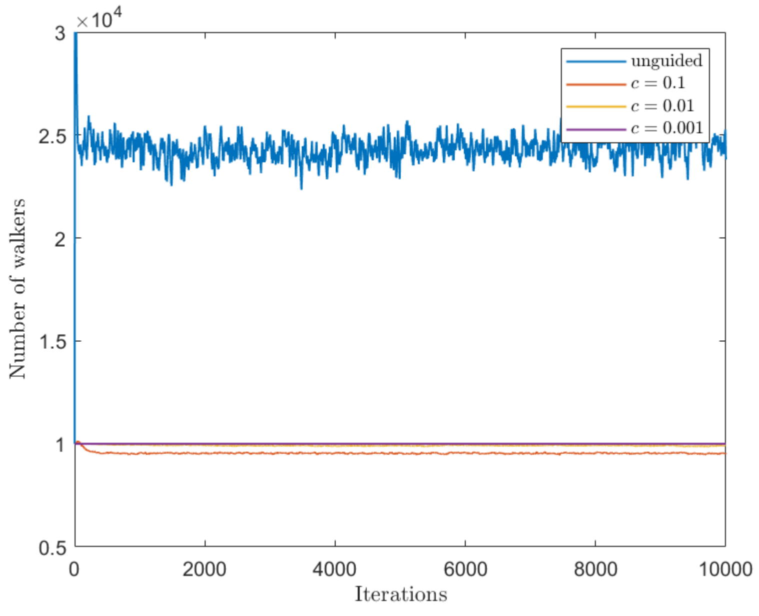

Figure 2: Fluctuations in the walker population during the simulation.

The number of walkers is also affected by the quality of the trial wave function. For a good trial wave function, the number of walkers remains constant over the entire simulation while for unguided DQMC, the walker population fluctuated quite a bit and is harder to control.
In conclusion, guided DQMC can give a significant accuracy boost if a good trial wave function is chosen. The choice of the trial wave function depends on the specific system under investigation.

3.2 Zero Point Energy of Ethane

In order to put the developed algorithm to use, we calculated the zero point energy (ZPE) of the ethane molecule (C_2H_6). We used unguided DQMC with the implementation of the split-join algorithm and (2.9) to update the trial energy after every iteration. As in [3], $\Delta t = 10$ and $\alpha = 0.01$ were used as simulation parameters. The simulation was carried out in Cartesian coordinates and therefore the position vector for each walker has a dimension of 24. To account for the different masses of the atoms, the random displacement of the atoms (line 5 and 6) used the width of the Gaussian distribution given by

$$\sigma_i = \sqrt{\frac{\Delta t}{m_i}}, \tag{3.6}$$

where m_i is the mass of the atom i. This this leads to the fact that heavier atoms get smaller random displacements than lighter atoms. To calculate the potential energies, we used DFTB+, a implementation of Density Functional based Tight Binding [6] with the Third-Order Parametrization for Organic and Biological Systems (3ob-3-1) parameter set [7]. This method is much faster than common density functional and configuration interaction methods and therefore allows for longer simulations with more walkers. After every move of a walker, DFTB+ was called via a Fortran subroutine and returned the potential energy $V(\boldsymbol{x})$ for the given coordinates of the walker.
The initial position of the walkers was also calculated with DFTB+ using a geometry optimization of the C_2H_6 molecule. This calculation also gave the minimal energy of the potential energy surface used to calculate the ZPE. 1000 walkers were then simulated from this initial position for 2000 time steps. For the calculation of the ZPE, the trial energy E_T was then averaged over all iterations after a short equilibration phase of 50 time steps. The ZPE is now calculated as the difference between the minimal energy of the potential energy surface and this averaged trial energy of the DQMC simulation. This calculation resulted in a ZPE of 0.07386 Hartree or 46.35 kcal/mol with a standard deviation of $2.362 \cdot 10^{-3}$ Hartree.
The calculation was preformed on sciCORE scientific computing center at University of Basel [8] and used about 14 hours of CPU time on a single core. In comparison, the ZPE calculated in [9] using more sophisticated methods gives a value of 46.39 kcal/mol.

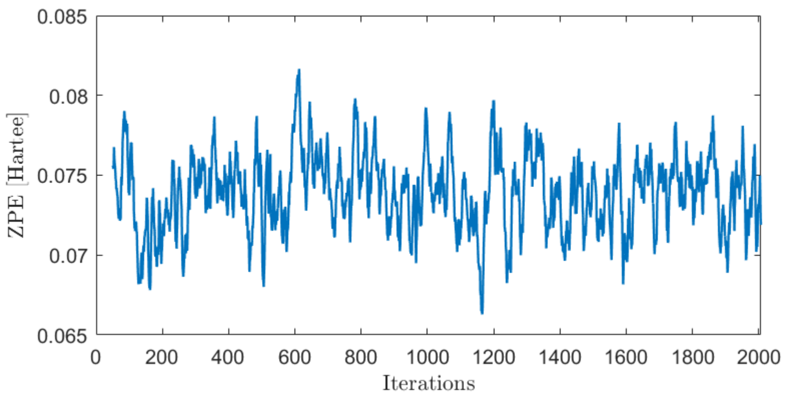

Figure 3: Fluctuations of the ZPE after the equilibration phase

The fluctuations of the ZPE are quite large. To decrease the fluctuations, one needs to do a simulation with more walkers and to improve the precision of the overall result, one can do a longer simulation or average the result over multiple simulations.

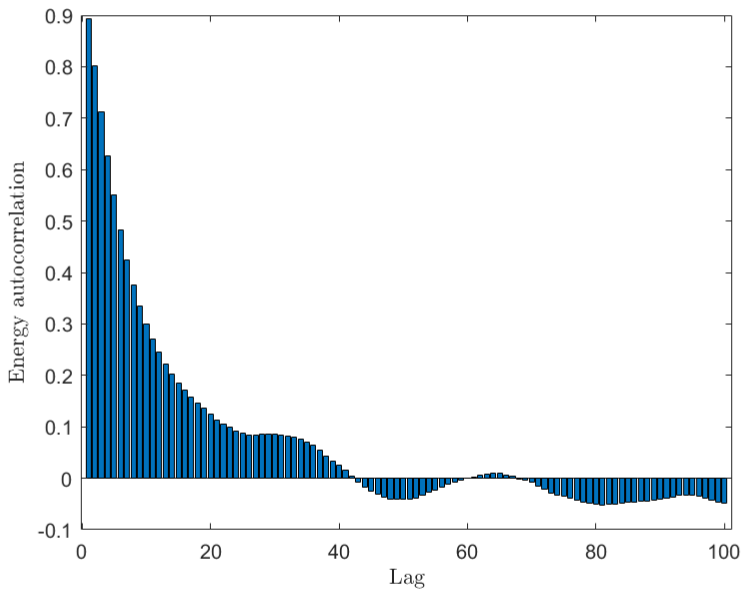

Figure 4: Autocorrelation function of the energy.

As a last investigation, we looked at the autocorrelation of the energy. Here we notice that after about 40 steps the energy becomes uncorrelated. This increases the confidence in our result and shows that the algorithm works well.

Chapter 4

Conclusions and Outlook

DQMC proved to be a simple and efficient tool to calculate ground state and zero point energies. The algorithm developed in this work can be applied to various molecules and quantum systems with little to no adjustments. For its simplicity and speed, the algorithm provides moderately accurate results. For more accurate results one can look at the methods and implementations pesented in Ref. [9]. The example calculation of the ZPE of ethane (C_2H_6) supports these claims as the result was fairly accurate using only little computational resources.
To further improve the algorithm, one can implement guided DQMC for molecules such as ethane. The challenge there is to find a good guiding wave function. This can be done using Gaussian guiding wave functions either in Cartesian coordinated or along the vibrational modes. This approach promises faster convergence, smaller fluctuations and therefore a more accurate result than the unguided DQMC algorithm.

Appendix A

The Box-Muller Algorithm

The Box-Muller algorithm transforms two $\mathcal{U}([0,1])$ random numbers (u_1, u_2) to two $\mathcal{N}(0,1)$ random numbers (x_1, x_2).

Algorithm 2 Box-Muller Algorithm

```
1: procedure BoxMuller($u_1, u_2$)
2:     $x_1 = sqrt(-2\log(u_1))\cos(2\pi u_2)$
3:     $x_2 = sqrt(-2\log(u_1))\sin(2\pi u_2)$
4: end procedure
```

Appendix B

The Metropolis Algorithm

The Metropolis algorithm generates random numbers distributed according to a (high-dimensional) probability distribution $p(\boldsymbol{x})$ using a Markov Chain. A Metropolis step generates a new random number $\boldsymbol{x}_{i+1}$ from a given $\boldsymbol{x}_i$ that follows the probability distribution $p(\boldsymbol{x})$.

Algorithm 3 Metropolis Step

```
 1: procedure METROPOLIS($\boldsymbol{x}_i$)
 2:     $\boldsymbol{x}' = f(\boldsymbol{x}_i, i, \text{"noise"})$        ▷ proposing a new random number
 3:     $\alpha = \frac{p(\boldsymbol{x}')}{p(\boldsymbol{x}_i)}$
 4:     if $\alpha \geq 1$ then
 5:         $\boldsymbol{x}_{i+1} = \boldsymbol{x}'$        ▷ Accepting the proposed number
 6:     else
 7:         draw $u \sim \mathcal{U}([0,1])$
 8:         if $u < \alpha$ then
 9:             $\boldsymbol{x}_{i+1} = \boldsymbol{x}'$        ▷ Accepting the proposed number
10:         else
11:             $\boldsymbol{x}_{i+1} = \boldsymbol{x}_i$        ▷ Rejecting the proposed number
12:         end if
13:     end if
14: end procedure
```

Appendix C

Numerical Derrivatives for $E_L(x)$ and $v(x)$

For a trial wave function $\psi(\boldsymbol{x})$ the first derivative used for the drift velocity $\boldsymbol{v}(\boldsymbol{x})$ is given by

$$\psi'(x) \approx \frac{\psi(x+h) - \psi(x-h)}{2h}, \tag{C.1}$$

and the second derivative used for the local energy $E_L(\boldsymbol{x})$ is given by

$$\psi''(x) \approx \frac{\psi(x+h) - 2\psi(x) + \psi(x-h)}{h^2}, \tag{C.2}$$

where h has to be chosen sufficiently small.

Bibliography

[1] D. Blackman and S. Vigna. Scrambled linear pseudorandom number generators, 2019.

[2] George H. Booth, Alex J. W. Thom, and Ali Alavi. Fermion monte carlo without fixed nodes: A game of life, death, and annihilation in slater determinant space. *The Journal of Chemical Physics*, 131(5):054106, 2009.

[3] Anne B. McCoy. Diffusion monte carlo approaches for investigating the structure and vibrational spectra of fluxional systems. *International Reviews in Physical Chemistry*, 25(1-2):77–107, 2006.

[4] Julien Toulouse, Roland Assaraf, and Cyrus J. Umrigar. Chapter fifteen - introduction to the variational and diffusion monte carlo methods. In Philip E. Hoggan and Telhat Ozdogan, editors, *Electron Correlation in Molecules – ab initio Beyond Gaussian Quantum Chemistry*, volume 73 of *Advances in Quantum Chemistry*, pages 285 – 314. Academic Press, 2016.

[5] C. J. Umrigar, M. P. Nightingale, and K. J. Runge. A diffusion monte carlo algorithm with very small time-step errors. *The Journal of Chemical Physics*, 99(4):2865–2890, 1993.

[6] B. Hourahine, B. Aradi, V. Blum, F. Bonafé, A. Buccheri, C. Camacho, C. Cevallos, M. Y. Deshaye, T. Dumitrică, A. Dominguez, S. Ehlert, M. Elstner, T. van der Heide, J. Hermann, S. Irle, J. J. Kranz, C. Köhler, T. Kowalczyk, T. Kubař, I. S. Lee, V. Lutsker, R. J. Maurer, S. K. Min, I. Mitchell, C. Negre, T. A. Niehaus, A. M. N. Niklasson, A. J. Page, A. Pecchia, G. Penazzi, M. P. Persson, J. Řezáč, C. G. Sánchez, M. Sternberg, M. Stöhr, F. Stuckenberg, A. Tkatchenko, V. W.-z. Yu, and T. Frauenheim. Dftb+, a software package for efficient approximate density functional theory based atomistic simulations. *The Journal of Chemical Physics*, 152(12):124101, 2020.

[7] M. Gaus, A. Goez, and M. Elstner. Parametrization and benchmark of dftb3 for organic molecules. *Journal of Chemical Theory and Compuation*, 9(1):338–354, 2013.

[8] `http://scicore.unibas.ch/`.

[9] Amir Karton, Branko Ruscic, and Jan M.L. Martin. Benchmark atomization energy of ethane: Importance of accurate zero-point vibrational energies and diagonal born–oppenheimer corrections for a 'simple' organic molecule. *Journal of Molecular Structure: THEOCHEM*, 811(1):345–353, 2007. Computational Organic Chemistry.