FYS4411 - Report on Project 1

Christian Fleischer

March 18, 2016

Abstract

The focus of this project is to do Variational Monte Carlo (VMC) studies of bosonic systems. The aim is to calculate the ground state energy of a trapped Bose gas for various numbers of particles in one, two and three dimensions, using the Variational Monte Carlo method. This is done both for systems with and without interaction between the bosons. The Metropolis algorithm is used with and without importance sampling, and the Steepest Descent method is used to optimize the variational parameters. The blocking method is used to do statistical analysis of the numerical data. Closed form expressions and other research are used to benchmark the results. For the non-interacting case the energies given by the closed form expression were reproduced exactly. For the interacting case the results were close to the benchmarks from other research when the system contained few particles, however, the error grew with increasing number of particles. This was due to an insufficient number of Monte Carlo cycles being used for large numbers of particles.

Contents

1 Introduction

The demonstration of Bose-Einstein condensation (BEC) in gases of alkali atoms confined in magnetic traps has significantly increased the interest in confined Bose systems. Some interesting aspects are the fraction of condensed atoms, the nature of the condensate, the excitations above the condensate and the critical temperature of BEC, as well as how the atomic density in the trap relates to the temperature.

One important feature of the trapped alkali and atomic hydrogen systems is that they are dilute, so the effective atom size is small in comparison to the trap size and the inter-atomic spacing. Many theoretical studies of these systems have been conducted in the framework of the Gross-Pitaevskii equation, and the key point for the validity of this description is that the average distance between atoms is significantly larger than the range of the inter-atomic interaction. In this case the physics is dominated by two-body collisions, which can be described in terms of the s-wave scattering length a. In defining the condition for diluteness, the crucial parameter is the gas parameter $x(\mathbf{r}) = n(\mathbf{r})a^3$, where $n(\mathbf{r})$ is the local density of the system. The Gross-Pitaevskii equation can be used for low values of the average gas parameter $x_{av} \leq 10^{-3}$, however, according to recent experiments the gas parameter may exceed this value due to the possible tuning of the scattering length in the presence of a Feshbach resonance. In this case, many-body Monte Carlo methods may be needed.

The aim of this project is to use the Variational Monte Carlo (VMC) method to calculate the ground state energy of a trapped Bose gas for various numbers of particles in one, two and three dimensions. To calculate the energy we use a specific trial wave function, and simulate using the VMC method with and without importance sampling. First we study the case where there is no interaction between the bosons. Then we include a correlation factor in the trial wave function, in order to study the interacting case where we have repulsion between the bosons. We have one variational parameter α, which is optimized using the Steepest Descent method. We also calculate the one-body density, and we do statistical analysis of the data using the blocking method. To benchmark our results we use closed form expressions and results from other research.

2 Methods

For the non-interacting case we use a spherical (S) harmonic trap, while for the interacting case we use an elliptical (E). We use traps in one, two and three dimensions and in the latter case the traps are given by

$$V_{ext}(\mathbf{r}) = \begin{cases} \frac{1}{2}m\omega_{ho}^2 r^2 & (S) \\ \frac{1}{2}m[\omega_{ho}^2(x^2+y^2) + \omega_z^2 z^2] & (E) \end{cases} \tag{1}$$

where ω_{ho} defines the trap potential strength. In the case of the elliptical trap ω_{ho} is the trap frequency in the perpendicular or xy plane and ω_z is the frequency in the z direction. The characteristic length of the trap is defined as $a_{ho} \equiv (\hbar/m\omega_{ho})^{\frac{1}{2}}$. The two-body Hamiltonian of the system is

$$H = \sum_i^N \left(\frac{-\hbar^2}{2m}\nabla_i^2 + V_{ext}(\mathbf{r}_i) \right) + \sum_{i<j}^N V_{int}(\mathbf{r}_i, \mathbf{r}_j), \tag{2}$$

We represent the inter-boson interaction by a pairwise, repulsive potential

$$V_{int}(|\mathbf{r}_i - \mathbf{r}_j|) = \begin{cases} \infty & |\mathbf{r}_i - \mathbf{r}_j| \leq a \\ 0 & |\mathbf{r}_i - \mathbf{r}_j| > a \end{cases} \tag{3}$$

where a is the hard-core diameter of the bosons. Our trial wave function for the ground state with N atoms is

$$\Psi_T(\mathbf{R}) = \Psi_T(\mathbf{r}_1, \mathbf{r}_2, \ldots \mathbf{r}_N, \alpha, \beta) = \prod_i g(\alpha, \beta, \mathbf{r}_i) \prod_{i<j} f(a, |\mathbf{r}_i - \mathbf{r}_j|), \tag{4}$$

where α and β are variational parameters. The single particle wave function is proportional to the harmonic oscillator function for the ground state, i.e.

$$g(\alpha, \beta, \mathbf{r}_i) = \exp\left[-\alpha(x_i^2 + y_i^2 + \beta z_i^2)\right]. \tag{5}$$

For spherical traps we have $\beta = 1$ and for non-interacting bosons ($a = 0$) we have $\alpha = 1/2a_{ho}^2$. The correlation wave function is

$$f(a, |\mathbf{r}_i - \mathbf{r}_j|) = \begin{cases} 0 & |\mathbf{r}_i - \mathbf{r}_j| \leq a \\ (1 - \frac{a}{|\mathbf{r}_i - \mathbf{r}_j|}) & |\mathbf{r}_i - \mathbf{r}_j| > a. \end{cases} \tag{6}$$

Throughout the project we use natural units, i.e. $\hbar = m = 1$.

2.1 Closed Form Expressions

In this project we want to find the expectation value of the local energy using the VMC method. In order to verify our implementation, we can test that the implementation reproduces results from using closed form expressions for the local energy. For our Hamiltonian H and trial wave function Ψ_T, we can find closed form expressions for the local energy given by

$$E_L(\mathbf{R}) = \frac{1}{\Psi_T(\mathbf{R})} H \Psi_T(\mathbf{R}). \tag{7}$$

Another benefit to having the closed form expressions is that in calculating the local energy we have to find the second derivative of Ψ_T, and doing this numerically can be computationally expensive. Therefore, we can find the expectation value of the local energy faster if we have a closed form expression for the local energy. The calculation of the closed form expressions for the non-interacting and interacting case can be found in the Appendix in Section 6.1.1 and Section 6.1.2. For the non-interacting case we have also included the calculation of the drift term used with importance sampling, while for the interacting case we have left this out (though it is implemented in the program).

2.2 Benchmarks for Verifying the Implementation

To verify the implementation further we compare our results with known benchmarks. For the non-interacting case the exact ground state energy and the corresponding wave function are known. For N particles in d dimensions we have

$$E = \frac{\omega d N}{2} \tag{8}$$

and

$$\Psi = \prod_{i=1}^{N} e^{-\frac{1}{2}r_i^2}. \tag{9}$$

In order for our trial wave function (Eq.(44)) to match the one for the exact ground state energy, we see that we need $\alpha = 0.5$. Using this α should allow our program to correctly reproduce the energies given by Eq.(8), provided that the implementation is correct.

For the interacting case the exact ground state energy is unknown. Instead we benchmark our results with other research. The results in Table 1 are for three dimensions and are provided by M. Hjorth-Jensen based on Ref. [1] (The article only provides the results for $N = 500$).

	$N = 10$	$N = 50$	$N = 100$
α	E	E	E
0.2	34.9	175	353
0.3	24.7	138	278
0.4	24.2	125	253
0.5	24.2	122	247
0.6	24.6	125	252
0.7	25.5	129	263

Table 1: Benchmarks of the energy E for the interacting system in three dimensions with N particles. The only variational parameter is α. The energy is minimal at $\alpha = 0.5$.

Another benchmark we can use for the interacting case is the one-body density given by the one-body density matrix, which has the form (Ref. [2])

$$\rho(\mathbf{r}', \mathbf{r}) = \langle \mathbf{\Psi}^\dagger(\mathbf{r}'), \mathbf{\Psi}(\mathbf{r}) \rangle. \tag{10}$$

We can find the one-body density by making a histogram of the sampled positions of the particles. The mean of the positions should be equal to 1 for the interacting case.

2.3 Variational Monte Carlo

The Variational Monte Carlo method we use in this project is based on the variational principle in quantum mechanics. The variational principle states that, given a Hamiltonian H and a trial wave function ψ_T, the expectation value $\langle H \rangle$ defined as

$$E[H] = \langle H \rangle = \frac{\int d\mathbf{R} \Psi_T^*(\mathbf{R}, \boldsymbol{\alpha}) H(\mathbf{R}) \Psi_T(\mathbf{R}, \boldsymbol{\alpha})}{\int d\mathbf{R} \Psi_T^*(\mathbf{R}, \boldsymbol{\alpha}) \Psi_T(\mathbf{R}, \boldsymbol{\alpha})} \tag{11}$$

is an upper bound to the ground state energy E_0 of the Hamiltonian, i.e.

$$E_0 \leq \langle H \rangle. \tag{12}$$

Here our trial wave function is dependant on some variational parameters α, and the goal of the VMC method is to vary these parameters until we find the lowest possible value of $\langle H \rangle$ in order to get an estimate for the ground state energy E_0. In general, the integrals we have to compute to find $\langle H \rangle$ are multi-dimensional ones, so using traditional integration methods like Gauss-Legendre is too computationally expensive. Therefore we turn to Monte Carlo methods.

2.3.1 Monte Carlo Integration with Brute Force Metropolis Sampling

This description of Monte Carlo integration and the Metropolis algorithm follows Ref. [3]. For a given trial wave function we define a probability distribution function (PDF)

$$P(\mathbf{R}) = \frac{|\Psi_T(\mathbf{R})|^2}{\int |\Psi_T(\mathbf{R})|^2 d\mathbf{R}} \tag{13}$$

Together with the local energy given in Eq.(7) we have that the approximation to the expectation value of the Hamiltonian is

$$\begin{aligned} E[H(\alpha)] =& \int P(\mathbf{R}) E_L(\mathbf{R}) d\mathbf{R} \\ \approx& \frac{1}{N} \sum_{i=1}^{N} P(\mathbf{R}_i, \boldsymbol{\alpha}) E_L(\mathbf{R}_i, \boldsymbol{\alpha}), \end{aligned} \tag{14}$$

where N is the number of Monte Carlo samples, $\mathbf{R_i}$ are the positions of the particles at step i and $\boldsymbol{\alpha}$ are the variational parameters.

The Metropolis algorithm is used to sample the probability distribution by a stochastic process. We define $\mathbf{P}_i^{(n)}$ to be the probability for finding the system in the state i at step n, and the algorithm is then

- Sample a possible new state j with some probability $T_{i\to j}$
- Accept the new state with probability $A_{i\to j}$ and use it as the next sample, or with probability $1 - A_{i\to j}$, reject the move and use the original state i as sample again.

We want to ensure that $\mathbf{P}_i^{(n\to\infty)} \to p_i$, so that regardless of the initial distribution, the method converges to the correct distribution. To ensure this we demand that the transition probability T and the acceptance probability A, fulfill the detailed balance requirement

$$\frac{A_{i\to j}}{A_{j\to i}} = \frac{p_i T_{i\to j}}{p_j T_{j\to i}}, \tag{15}$$

where $p_i = P(\mathbf{R}_i)$ and $p_j = P(\mathbf{R}_j)$. The Metropolis algorithm then uses the following ratio of probabilities to determine whether or not to accept a move

$$\frac{p_j}{p_i} = \frac{T_{i\to j} A_{i\to j}}{T_{j\to i} A_{j\to i}} \tag{16}$$

When using the Metropolis algorithm we can either use brute force or importance sampling. If we use the brute force Metropolis algorithm, we assume that $T_{i\to j} = T_{j\to i}$. Then the ratio used by the Metropolis algorithm is only dependant on the acceptance probabilities, i.e. the ratio is given by

$$w = \frac{p_j}{p_i} = \frac{A_{i\to j}}{A_{j\to i}} = \frac{|\Psi_T(\mathbf{R}_j)|^2}{|\Psi_T(\mathbf{R}_i)|^2} \tag{17}$$

The algorithm for estimating the ground state energy given a set of variational parameters α is then

- Fix the number of Monte Carlo steps and create an initial state $\mathbf{R}$ using the given variational parameters $\boldsymbol{\alpha}$. Set a step size $\Delta\mathbf{R}$ and calculate $|\Psi_T^{\alpha}(\mathbf{R})|^{\mathbf{2}}$.

- Initialize the local energy and start the Monte Carlo calculations.
  - Choose a random particle and update its position in order to create a trial state: $\mathbf{R}_p = \mathbf{R} + r\Delta\mathbf{R}$ where r is a random variable $r \in [0, 1]$
  - Calculate $|\Psi_T^\alpha(\mathbf{R_p})|^\mathbf{2}$ and use the Metropolis algorithm to accept or reject the move by calculating the ratio w. If $w \geq s$, where s is a random number $s \in [0, 1]$, the new state is accepted. Otherwise we keep the old state.
  - If the new state is accepted, then we set $\mathbf{R} = \mathbf{R}_p$.
  - Update the local energy.
- Finish and compute the final estimate for the ground state energy.

Algorithm for one Brute Force Metropolis step in C++

```
// Evaluate the wave function for current positions
double waveFunctionOld = m_waveFunction->evaluate(m_particles);

// Choose a random particle and change its position by a random amount
    creating a trial state
int randomParticle = Random::nextInt(m_numberOfParticles);
std::vector<double> positionChange(m_numberOfDimensions);

for (int i=0; i<m_numberOfDimensions; i++){
    positionChange[i] = (Random::nextDouble()*2-1)*m_stepLength;
    m_particles[randomParticle]->adjustPosition(positionChange[i], i);
}

// Evaluate the wave function for the trial state
double waveFunctionNew = m_waveFunction->evaluate(m_particles);
// Metropolis ratio
double qratio = waveFunctionNew*waveFunctionNew / (waveFunctionOld*
    waveFunctionOld);

// Check if trial state is accepted
if (Random::nextDouble() <= qratio){
    return true;
}

for (int i=0; i<m_numberOfDimensions; i++){
    // If trial state is not accepted, revert to old position for chosen
        particle (revert to old state)
    m_particles[randomParticle]->adjustPosition(-positionChange[i], i);
}

return false;
```

2.3.2 Metropolis-Hastings Algorithm (Importance Sampling)

For the description of the Metropolis-Hastings Algorithm we again follow Ref. [3]. When using importance sampling the walk in coordinate space is biased by the trial wave function, so the

walker is more likely to move towards regions where the trial wave function is large. The trajectory in coordinate space is generated by the Fokker-Planck equation and the Langevin equation. To find the new positions in coordinate space we solve the Langevin equation using Euler's method. The Langevin equation

$$\frac{\partial x(t)}{\partial t} = DF(x(t)) + \eta, \tag{18}$$

where η is a random variable and D is the diffusion coefficient, gives the new position

$$y = x + DF(x)\Delta t + \xi\sqrt{\Delta t}, \tag{19}$$

where ξ is gaussian random variable, Δt is a chosen time step and $F(x)$ is the function responsible for drifting the walker towards regions where the wave function is large. Δt is treated as a parameter and $\Delta t \in [0.001, 0.01]$ generally yields fairly stable values of the ground state energy. The diffusion coefficient D is equal to $1/2$ and comes from the $1/2$ factor in the kinetic energy operator. The Fokker-Planck equation describes the process of isotropic diffusion characterized by a time-dependent probability density $P(x,t)$ and is given by

$$\frac{\partial P}{\partial t} = \sum_i D\frac{\partial}{\partial \mathbf{x}_i}\left(\frac{\partial}{\partial \mathbf{x}_i} - \mathbf{F}_i\right)P(\mathbf{x},t), \tag{20}$$

where $\mathbf{F_i}$ is the i^{th} component of the drift term. By setting the left hand side to zero we obtain the convergence to a stationary probability density. For the resulting equation to be satisfied all the terms of the sum have to be equal to zero, i.e.

$$\frac{\partial^2 P}{\partial \mathbf{x}_i^2} = P\frac{\partial}{\partial \mathbf{x}_i}\mathbf{F}_i + \mathbf{F}_i\frac{\partial}{\partial \mathbf{x}_i}P. \tag{21}$$

The drift vector $\mathbf{F}$ should be on the form $\mathbf{F} = g(\mathbf{x})\frac{\partial P}{\partial \mathbf{x}}$, which gives us

$$\frac{\partial^2 P}{\partial \mathbf{x}_i^2} = P\frac{\partial g}{\partial P}\left(\frac{\partial P}{\partial \mathbf{x}_i}\right)^2 + Pg\frac{\partial^2 P}{\partial \mathbf{x}_i^2} + g\left(\frac{\partial P}{\partial \mathbf{x}_i}\right)^2. \tag{22}$$

The condition of stationary density requires that the terms cancel each other out, and that is only possible if $g = \frac{1}{P}$. This leads to

$$\mathbf{F} = 2\frac{1}{\Psi_T}\nabla\Psi_T, \tag{23}$$

which is the so-called *quantum force*. By pushing the walker towards regions where the trial wave function is large, this term increases the efficiency of the simulation compared to the brute force Metropolis algorithm where the walker has the same probability of moving in every direction.

From the Fokker-Planck equation we get a transition probability given by the Green's function

$$G(y,x,\Delta t) = \frac{1}{(4\pi D\Delta t)^{3N/2}}\exp(-(y - x - D\Delta t F(x))^2/4D\Delta t), \tag{24}$$

which means that the ratio used in the Metropolis algorithm

$$w = \frac{|\Psi_T(\mathbf{R}_j)|^2}{|\Psi_T(\mathbf{R}_i)|^2} = q(y,x) = \frac{|\Psi_T(y)|^2}{|\Psi_T(x)|^2}, \tag{25}$$

is now replaced by

$$q(y,x) = \frac{G(x,y,\Delta t)|\Psi_T(y)|^2}{G(y,x,\Delta t)|\Psi_T(x)|^2}. \tag{26}$$

Using this ratio and Eq.(19), the algorithm is called the Metropolis-Hastings algorithm.

2.3.3 Blocking

Here follows a brief explanation of the blocking method based on Ref. [3]. For a more detailed explanation consult the reference. At each Monte Carlo step in our simulation we sample a local energy and the mean of these samples $\langle H \rangle$ is our estimate for the ground state energy. If we assume that the n samples are uncorrelated our best estimate for the standard deviation of the mean $\langle H \rangle$ is given by

$$\sigma = \sqrt{\frac{1}{n}(\langle H^2 \rangle - \langle H \rangle^2)}. \tag{27}$$

However, this is a too optimistic estimate of the error in our calculations because the samples are correlated. Therefore we need to rewrite our expression for the standard deviation to

$$\sigma = \sqrt{\frac{1+2\tau/\Delta t}{n}(\langle H^2 \rangle - \langle H \rangle^2)}, \tag{28}$$

where τ is the correlation time, i.e. the time between a given sample and the next uncorrelated sample. Δt is the time between each sample. If $\Delta t \gg \tau$ the estimate in Eq.(27) still holds, however, usually $\Delta t < \tau$. When using the blocking method we divide the sequence of samples into blocks, and then calculate the mean and variance of each block separately. Finally we calculate the total mean and variance of all of the blocks. The size of the blocks has to be large enough that sample j of block i is not correlated with sample j of block $i+1$. For this, the correlation time τ would be a good choice, however, τ is too expensive to compute.

Instead we can plot the standard deviation as a function of block size. As long as the block size is so small that the blocks are correlated the standard deviation will increase with increasing block size. However, once the block size is large enough that the blocks are uncorrelated, we reach a plateau. Therefore, when the standard deviation stops increasing, the plateau value of the standard deviation will be a good estimate of the error in our results.

Blocking in Python

```
import numpy as np
import matplotlib.pyplot as plt

def readData(filename):

    infile = open("Data/%s" %filename, 'r')
    energies = []

    for line in infile:
        energies.append(float(line))

    infile.close()
    return np.asarray(energies)

def blocking(energies, nBlocks, blockSize):
```

```
    meansOfBlocks = np.zeros(nBlocks)

    for i in range(nBlocks):
        energiesOfBlock = energies[i*blockSize:(i+1)*blockSize]
        meansOfBlocks[i] = sum(energiesOfBlock)/blockSize

    mean = sum(meansOfBlocks)/nBlocks
    mean2 = sum(meansOfBlocks**2)/nBlocks
    variance = mean2 - mean**2

    return mean, variance

if __name__ == "__main__":
    N = 1
    energies = readData("energiesN%i.dat" %N)

    deltaBlockSize = 100
    minBlockSize = 10
    maxBlockSize = 10000
    numberOfSizes = (maxBlockSize-minBlockSize)/deltaBlockSize + 1
    largestBlockSize = minBlockSize + (numberOfSizes-1)*deltaBlockSize #
        9910

    #blockSizes = np.zeros(numberOfSizes)
    blockSizes = np.linspace(minBlockSize, largestBlockSize,
        numberOfSizes).astype(int)
    blockAmounts = len(energies)/blockSizes#np.zeros(numberOfSizes)
    means = np.zeros(numberOfSizes)
    variances = np.zeros(numberOfSizes)

    for i in range(numberOfSizes):
        #blockSize = minBlockSize + i*deltaBlockSize
        #blockAmount = len(energies)/blockSize
        #mean, variance = blocking(energies, blockAmount, blockSize)
        mean, variance = blocking(energies, blockAmounts[i], blockSizes[i
            ])
        means[i] = mean
        variances[i] = variance

    standardDeviation = np.sqrt(abs(variances)/(blockAmounts-1.))
    plt.plot(blockSizes, standardDeviation)
    plt.xlabel("Block Size")
    plt.ylabel(r"Standard Deviation $\sigma$")
    plt.title("N=%i" %N)
    plt.ticklabel_format(style='sci', axis='y', scilimits=(0,0))
    plt.show()
```

2.3.4 The Steepest Descent Method (Gradient Descent)

We want to optimize the variational parameters to minimize the estimation of the ground state energy. To do this we can use the Conjugate Gradient method or the Steepest Descent

method. In this project we have chosen the simpler Steepest Descent method and we limit the problem to only one variational parameter α. To optimize α we treat our estimate to the ground state energy as a function of α. The Steepest Descent method then finds a local minimum for this function by iteratively moving in the direction given by the negative of the gradient of the function at the current point (Ref. [5]). At each step the move is proportional to a chosen step length γ, and one step is then

$$\alpha_{i+1} = \alpha_i - \gamma \bar{E}_\alpha, \tag{29}$$

where

$$\bar{E}_\alpha = \frac{d\langle E_L[\alpha]\rangle}{d\alpha} \tag{30}$$

is the gradient. We then have that $E_L[\alpha_{i+1}] \leq E_L[\alpha_i]$. In order to find $\bar{E}_\alpha$ we also need the derivative of the trial wave function, which we define as

$$\bar{\psi}_\alpha = \frac{d\psi[\alpha]}{d\alpha}. \tag{31}$$

By using the chain rule and the hermiticity of the Hamiltonian we get an expression for the gradient $\bar{E}_\alpha$ (Ref. [4])

$$\bar{E}_\alpha = 2\left(\left\langle \frac{\bar{\psi}_\alpha}{\psi[\alpha]} E_L[\alpha]\right\rangle - \left\langle \frac{\bar{\psi}_\alpha}{\psi[\alpha]}\right\rangle \langle E_L[\alpha]\rangle\right), \tag{32}$$

so we need to compute

$$\left\langle \frac{\bar{\psi}_\alpha}{\psi[\alpha]} E_L[\alpha]\right\rangle, \tag{33}$$

and

$$\left\langle \frac{\bar{\psi}_\alpha}{\psi[\alpha]}\right\rangle, \tag{34}$$

in addition to $\langle E_L[\alpha]\rangle$.

To optimize α we guess an initial value α. Then, using few Monte Carlo cycles, we run the simulation and sample the expectation values Eq.(33), Eq.(34) and the expectation value for the ground state energy (as usual). Using the expectation values we calculate the gradient Eq.(32) and find a new α using Eq.(29). We repeat this until we have found an optimal α to some desired precision. Finally, we do a large-scale Monte Carlo simulation (many cycles) using the optimal α to find a good estimate to the ground state energy.

Steepest Descent Method in C++

```
int iteration = 0;  // Count iterations so we can force quit after a
    given number max iterations
double alphaNew = alpha;

do{
    alpha = alphaNew;    // Update alpha

    // Run Monte Carlo simulation to find expectation values
    m_system->getInitialState()->setupInitialState();
    m_system->getWaveFunction()->adjustParameter(alpha, 0);
```

```
    m_system->runMetropolisSteps(numberOfMetropolisSteps,
        importanceSampling, false, false, false, false);

    double derivative;  //derivative of local energy.
    // Expectation values needed to calculate derivative of local energy:
    double energy = m_system->getSampler()->getEnergy();
    double waveFuncEnergy = m_system->getSampler()->getWaveFuncEnergy();
    double waveFuncDerivative = m_system->getSampler()->
        getWaveFuncDerivative();

    derivative = 2*(waveFuncEnergy - energy*waveFuncDerivative);
    // Find new alpha
    alphaNew = alpha - derivative*m_stepLengthSD;
    iteration++;
    cout << "Iterations: " << iteration << endl;
    cout << "Alpha: " << alpha << "\033[F";

}while(abs(alphaNew - alpha) > tol && iteration < maxIterations);
// Loop ends when requested tolerance for optimal alpha has been reached
    or after max iterations.

cout << "Total iterations: " << iteration << endl;
const char* message = "Optimal alpha: ";
if (iteration==maxIterations) message = "Max iterations reached.\nAlpha
    at max iterations: ";
cout << message << alpha << endl;

// Performing large MC simulation with optimal aplha:
m_system->getInitialState()->setupInitialState();
m_system->getWaveFunction()->adjustParameter(alpha, 0);
m_system->runMetropolisSteps((int) 1e6, importanceSampling, false, false,
    true, true);
```

3 Results and Discussion

3.1 Non-Interacting Case with Brute Force

With 10^4 Monte Carlo cycles, $\alpha = 0.5$, step length 0.1 and using brute force, we get the results listed in Table 2 for analytical kinetic energy, and the results in Table 3 for numerical kinetic energy.

From Table 2 and Table 3 we see that using the analytical kinetic energy and using the numerical kinetic energy gives the same answer for $N = 1, 10, 100, 500$ and $D = 1, 2, 3$. The results are also consistent with the exact ground state energy given by Eq.(8), so the implementation is working as intended in this case. We also see that using analytical kinetic energy reduces CPU time significantly as expected, since differentiating numerically is computationally expensive.

N	D	E	σ^2	A. Rate	CP Time [s]
1	1	0.5	0	0.970663	0.013514
10	1	5	0	0.968774	0.040288
100	1	50	0	0.971552	0.229013
500	1	250	0	0.975108	0.931867
1	2	1	0	0.958551	0.01321
10	2	10	0	0.955662	0.048426
100	2	100	0	0.963996	0.237468
500	2	500	0	0.963774	1.11984
1	3	1.5	0	0.950106	0.010382
10	3	15	0	0.94266	0.047171
100	3	150	0	0.952217	0.23228
500	3	750	0	0.954662	0.982195

Table 2: Estimates to the ground state energy E using an analytical expression for the kinetic energy. 10^4 MC cycles and the brute force Metropolis algorithm were used and the only variational parameter was $\alpha = 0.5$. N is the number of particles, D is the number of dimensions and A. Rate is the acceptance rate. σ^2 is the variance when (wrongly) assuming uncorrelated samples.

N	D	E	σ^2	A. Rate	CP Time [s]
1	1	0.5	-3.60822e-16	0.970663	0.018768
10	1	5	-5.68434e-14	0.968774	0.136459
100	1	50	1.05956e-10	0.971552	6.06854
500	1	250	3.50556e-08	0.975108	142.024
1	2	1	1.86517e-14	0.958551	0.020823
10	2	10	-3.55271e-13	0.955662	0.196212
100	2	100	1.47884e-09	0.963996	12.1435
500	2	500	5.69125e-07	0.963774	288.079
1	3	1.5	-1.59872e-14	0.950106	0.023083
10	3	15	2.81375e-12	0.94266	0.290512
100	3	150	4.48563e-09	0.952217	18.5267
500	3	750	2.82086e-06	0.954662	436.082

Table 3: Estimates to the ground state energy E using numerical differentiation for the kinetic energy. 10^4 MC cycles and the brute force Metropolis algorithm were used and the only variational parameter was $\alpha = 0.5$. N is the number of particles, D is the number of dimensions and A. Rate is the acceptance rate. σ^2 is the variance when (wrongly) assuming uncorrelated samples.

3.2 Non-Interacting Case with Importance Sampling

With 10^4 Monte Carlo cycles, $\alpha = 0.5$, step length 0.1 and using importance sampling, we get the results listed in Table 4 for analytical kinetic energy, and the results in Table 5 for numerical kinetic energy.

From Table 4 and Table 5 we see that when using importance sampling we get the same energies as we did when using brute force, both for analytical kinetic energy and numerical

N	D	E	σ^2	A. Rate	CP Time [s]
1	1	0.5	0	0.954995	0.027817
10	1	5	0	0.950106	0.063358
100	1	50	0	0.960996	0.346255
500	1	250	0	0.957217	1.47094
1	2	1	0	0.927659	0.059337
10	2	10	0	0.924547	0.169605
100	2	100	0	0.936882	1.0398
500	2	500	0	0.937993	4.77655
1	3	1.5	0	0.917213	0.097335
10	3	15	0	0.908879	0.382984
100	3	150	0	0.920436	2.98585
500	3	750	0	0.92088	13.8103

Table 4: Estimates to the ground state energy E using an analytical expression for the kinetic energy. 10^4 MC cycles and the importance sampling Metropolis-Hastings algorithm were used. The only variational parameter was $\alpha = 0.5$. N is the number of particles, D is the number of dimensions and A. Rate is the acceptance rate. σ^2 is the variance when (wrongly) assuming uncorrelated samples. The acceptance rate is generally lower than when using the brute force Metropolis algorithm.

N	D	E	σ^2	A. Rate	CP Time [s]
1	1	0.5	2.44249e-15	0.954995	0.033533
10	1	5	3.23297e-13	0.950106	0.141288
100	1	50	7.36691e-11	0.960996	6.15015
500	1	250	2.52985e-08	0.957217	141.466
1	2	1	2.44249e-15	0.927659	0.066272
10	2	10	1.1795e-12	0.924547	0.315584
100	2	100	2.07001e-09	0.936882	12.9807
500	2	500	7.03323e-07	0.937993	292.192
1	3	1.5	-3.55271e-15	0.917213	0.113486
10	3	15	1.9611e-12	0.908879	0.609458
100	3	150	7.58519e-09	0.920436	20.6846
500	3	750	5.58805e-06	0.92088	458.789

Table 5: Estimates to the ground state energy E using numerical differentiation for the kinetic energy. 10^4 MC cycles and the importance sampling Metropolis-Hastings algorithm were used. The only variational parameter was $\alpha = 0.5$. N is the number of particles, D is the number of dimensions and A. Rate is the acceptance rate. σ^2 is the variance when (wrongly) assuming uncorrelated samples. The acceptance rate is generally lower than when using the brute force Metropolis algorithm.

kinetic energy. The acceptance rate is generally lower when using importance sampling, which may be due to the system reaching its most likely state in fewer Monte Carlo cycles, and once the most likely state has been reached the system is less likely to accept a move to another state. Finally, we see that using importance sampling increases CPU time somewhat, which

makes sense since the importance sampling requires some additional calculations to be made.

We also look at how the time step used in importance sampling affects the result. The results are in Table 6 and Table 7 for three dimensions, 10 particles and 10^4 Monte Carlo cycles.

dt	E	σ^2	A. Rate	CP Time [s]
0.1	15	0	0.725858	0.391393
0.01	15	0	0.908879	0.386808
0.001	15	0	0.974108	0.391305

Table 6: Results for different time step values dt used in the Metropolis-Hastings algorithm. Only the acceptance rate, A. Rate, changes significantly for different dt. An analytical expression for the kinetic energy was used to get these results.

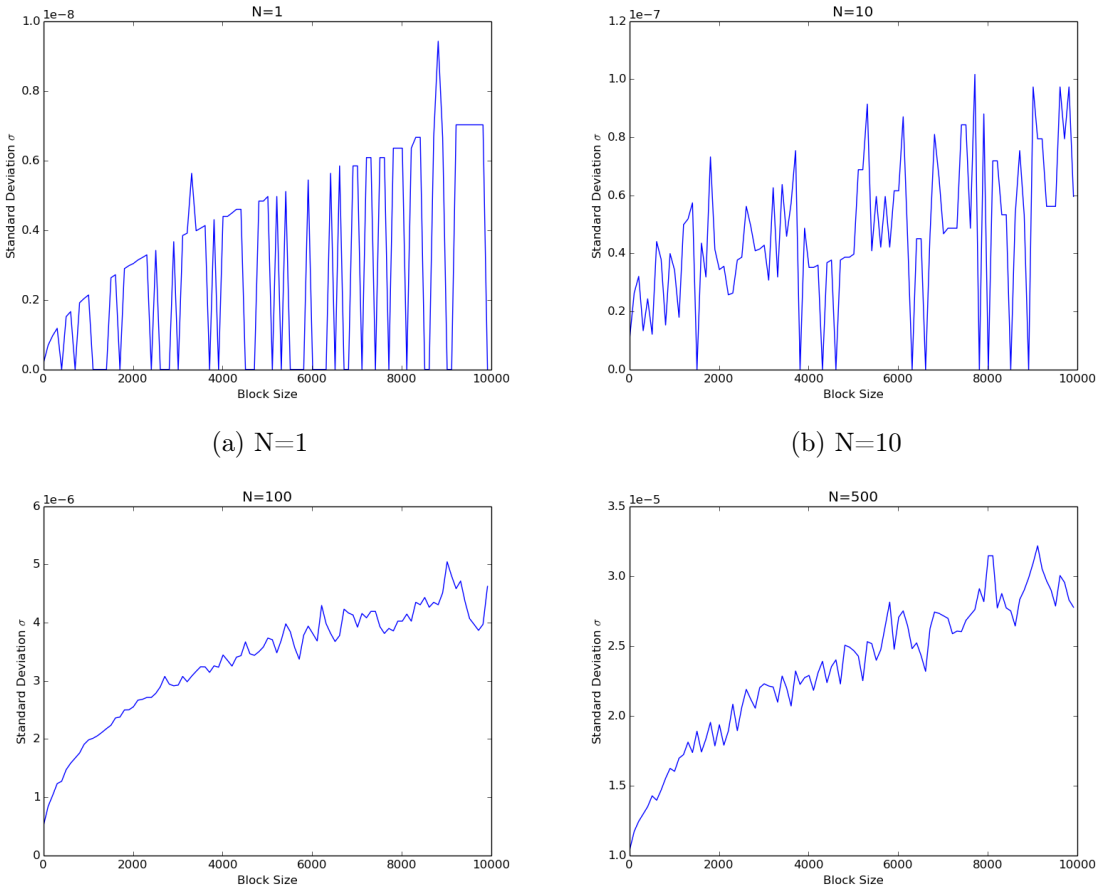
dt	E	σ^2	A. Rate	CP Time [s]
0.1	15	1.9611e-12	0.725858	0.623865
0.01	15	7.67386e-13	0.908879	0.605917
0.001	15	1.62004e-12	0.974108	0.61988

Table 7: Results for different time step values dt used in the Metropolis-Hastings algorithm. Only the acceptance rate, A. Rate, changes significantly for different dt. Numerical differentiation to find the kinetic energy was used to get these results.

From Table 6 and Table 7 we see that only the acceptance rate changes significantly as a function of the time step dt. The acceptance ratio increases with decreasing dt.

3.3 Statistical Analysis with Blocking

In order to do proper statistical analysis of our results we use the blocking method. Figure 1 provides the blocking results for a system using imprtance sampling and with 10^5 Monte Carlo cycles, $\alpha = 0.5$, step length 0.1 and $N = 1, 10, 100, 500$ in three dimensions.



(a) N=1

(b) N=10

(c) N=100

(d) N=500

Figure 1: The figures show the standard deviation for a set of energy samples using the blocking method with different block sizes. 1a and 1b show that for $N = 1$ and $N = 10$ it's hard to recognize the curve we expect from blocking. 1c and 1d show that for $N = 100$ and $N = 500$ the shape of the curve is closer to what we expect and it seems like the standard deviation reaches a plateau at around $\sigma = 4.5 \times 10^{-6}$ for $N = 100$ and around $\sigma = 3.0 \times 10^{-5}$ for $N = 500$.

From Figure 1 we see that for $N = 1$ and $N = 10$ it's hard to recognize the curve we expect from blocking, though there may be a plateau for the standard deviation at around $\sigma = 0.7 \times 10^{-8}$ for $N = 1$ and around $\sigma = 0.8 \times 10^{-7}$ for $N = 10$. For $N = 100$ and $N = 500$ the shape of the curve is closer to what we expect and it seems like the standard deviation reaches a plateau at around $\sigma = 4.5 \times 10^{-6}$ for $N = 100$ and around $\sigma = 3.0 \times 10^{-5}$ for $N = 500$. To be able to see the plateau more clearly for $N = 100$ and $N = 500$ we could increase the maximum block size, however, that would also require an increase in the number

of Monte Carlo cycles, which would be too expensive to compute with our implementation.

3.4 Interacting Case with Brute Force

For the interacting case we want to see what energies we get for different values of the variational parameter α, in order to try to find a minimum. Table 8 contains the results using analytical kinetic energy, while Table 9 contains the results using numerical kinetic energy. We used 10^4 MC steps, step length 0.1, $N = 10, 50, 100$ and the brute force Metropolis algorithm.

	$N = 10$	$N = 50$	$N = 100$
α	E	E	E
0.2	32.0767	150.252	276.176
0.3	26.5251	145.337	276.827
0.4	24.5062	130.189	296.634
0.5	24.4397	128.498	295.247
0.6	25.0616	142.784	288.364
0.7	26.0932	131.617	289.003

Table 8: Estimates E to the ground state energy for different values of the variational parameter α using 10^4 MC cycles. N is the number of particles and the number of dimensions is three. We see that for $N = 10$ the energy has a minimum for $\alpha = 0.5$. Here an analytical expression for the kinetic energy was used.

	$N = 10$	$N = 50$	$N = 100$
α	E	E	E
0.2	32.0629	136.58	267.122
0.3	26.5864	132.5	270.454
0.4	24.5607	129.259	273.993
0.5	24.4076	129.624	275.142
0.6	25.1516	130.912	275.94
0.7	26.1007	133.409	275.884

Table 9: Estimates E to the ground state energy for different values of the variational parameter α using 10^4 MC cycles. N is the number of particles and the number of dimensions is three. We see that for $N = 10$ the energy has a minimum for $\alpha = 0.5$. Here numerical differentiation was used to find the kinetic energy.

We see from Tables 8 and 9 that for $N = 10$ using the analytical and numerical kinetic energies give about the same estimate to the ground state energy, and the values we get are also fairly consistent with the benchmark values in Table 1. We also see that for $N = 10$ the energy is minimal for $\alpha = 0.5$. When comparing to the non-interacting case, we see that for $N = 10$ particles in three dimensions with $\alpha = 0.5$, the energy increases from 15 to about 24.4 when we include the correlation. As we increase the number of particles N however, the differences become much larger and $\alpha = 0.5$ no longer gives minimal energy. A possible reason for this is that as the number of particles increases, 10^4 MC cycles is no longer enough to give accurate results. There could also be something wrong with the implementation. In order to check if the number of MC cycles is the problem, we run the simulations again with

10^6 MC cycles, only for the analytical kinetic energy (numerical differentiation would be too computationally expensive in this case). The results are given in Table 10.

	$N = 10$	$N = 50$	$N = 100$
α	E	E	E
0.2	35.8434	180.893	378.232
0.3	27.8286	143.203	298.536
0.4	25.025	132.897	277.846
0.5	24.3884	127.505	265.032
0.6	25.1424	130.953	331.251
0.7	25.7855	136.821	289.083

Table 10: Estimates E to the ground state energy for different values of the variational parameter α using 10^6 MC cycles. N is the number of particles and the number of dimensions is three. We see that for N values the energy has a minimum for $\alpha = 0.5$. All the values are also more consistent with the benchmark then they were with 10^4 cycles.. Here an analytical expression for the kinetic energy was used.

From Table 10 we see that increasing the number of MC cycles indeed made the results more consistent with the benchmarks in 1, and for all values of N the energy has a minimum for $\alpha = 0.5$. However, $N = 100$ and $\alpha = 0.6$ still gives an inconsistent result, so the results should be further improvable by increasing the number of MC cycles even more.

3.5 Optimizing the Variational Parameter with Steepest Descent

We used the Steepest Descent method for a system in three dimensions with $N = 10$, using the brute force Metropolis algorithm and 10^5 MC cycles per step in the optimization, to find the optimal α. Starting from $\alpha = 0.7$ the program gave an optimal $\alpha = 0.497908$ after 9 steps. After finding the optimal α the program did another Monte Carlo simulation to find the ground state energy using the optimal α and 10^6 MC cycles. The resulting estimate for the ground state energy is given in Table 11.

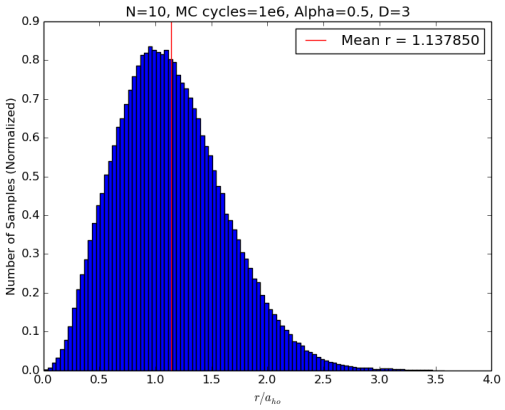
α	E
0.497908	24.3999

Table 11: Estimate to the ground state energy for a three dimensional interacting system with $N = 10$ particles with the variational parameter α. Numerical differentiation was used to find the kinetic energy.

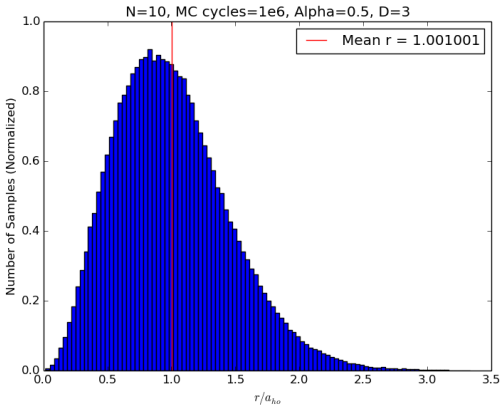
3.6 One-body Density

For $N = 10$ particles and 10^6 MC cycles in three dimensions with $\alpha = 0.5$, the results for the one-body density are given in Figure 2.

From Figure 2 we see that the distribution for the non-interacting case is shifted more to the right and has a greater mean value. For the interacting case the mean value is about 1 as expected from the benchmark. The difference in the mean value is not very large, however it is large enough that the correlations induced by the Jastrow (correlation) factor are significant.



(a) Non-interacting case



(b) Interacting case

Figure 2: Histograms for one-body density for the non-interacting case 2a and the interacting case 2b. The histograms show the distributions of sampled positions. The distribution in 2a is shifted further to the right and has a greater mean value. The mean in 2b is about 1 as expected, and we see that the difference in mean value is not that large, but large enough to be significant. N is the number of particles, Alpha is the variational parameter and D is the number of dimensions. An analytical expression for the kinetic energy was used in both cases.

4 Conclusion

Throughout this project we have seen that for a spherical harmonic oscillator with no interaction we can estimate the ground state energy exactly using a closed form expression for the kinetic energy. When using numerical differentiation for the kinetic energy there is a small, insignificant error because the method used to differentiate introduces a numerical error. We have also seen that the importance sampling Metropolis-Hastings algorithm has a lower acceptance rate than the brute force Metropolis algorithm, probably because the Metropolis-Hastings algorithm reaches the most likely state in fewer MC cycles. Finally we have seen that using an optimal value for the variational parameter α gives the following estimate to the ground state energy; $E = 24.3999$ for an interacting system in three dimensions with $N = 10$ particles, which is close to a benchmark provided by other research. The corresponding ground state energy for a non-interacting system is $E = 15$.

The optimization of α can be improved by using more advanced methods, like the Conjugate Gradient method, instead of the Steepest Descent method. The program can be made more efficient by using parallelization. Also, by doing some modifications to the program, it can be used to study fermionic systems as well.

5 References

References

[1] J. K. Nilsen, J. Mur-Petit, M. Guilleumas, M. Hjorth-Jensen, and A. Polls, *Vortices in atomic Bose-Einstein condensates in the large-gas-parameter region*, Phys. Rev. A **71**, 053610 (2005).

[2] J. L. DuBois and H. R. Glyde, H. R., *Bose-Einstein condensation in trapped bosons: A variational Monte Carlo analysis*, Phys. Rev. A **63**, 023602 (2001).

[3] M. Hjorth-Jensen, *Computational Physics 2: Variational Monte Carlo methods*, `https://github.com/CompPhysics/ComputationalPhysics2/blob/master/doc/pub/vmc/pdf/vmc-print.pdf`, Read: 17.03.2016.

[4] M. Hjorth-Jensen, *Conjugate gradient methods and other optimization methods*, `https://github.com/CompPhysics/ComputationalPhysics2/blob/master/doc/pub/cg/pdf/cg-print.pdf`, Read: 17.03.2016.

[5] `https://en.wikipedia.org/wiki/Gradient_descent`, Read: 17.03.2016.

6 Appendix

URL to github folder with source files etc.
`https://github.com/christianfleischer/FYS4411_Projects/tree/master/Project1`

`SSH URL to github repository for all FYS4150 projects:`
`git@github.com:christianfleischer/FYS4411_Projects.git`

6.1 Calculations of Closed Form Expressions

6.1.1 Without Interaction

For the spherical non-interacting case we have only the spherical harmonic oscillator potential, i.e. we set $a = 0$ and $\beta = 1$. Our single particle wave function in one dimension is

$$\Psi_T(x) = e^{-\alpha x^2} \tag{35}$$

and the corresponding Hamiltonian is

$$H(x) = -\frac{1}{2}\frac{\partial^2}{\partial x^2} + \frac{1}{2}\omega^2 x^2 \tag{36}$$

The derivative of ψ_T is

$$\frac{\partial \Psi_T}{\partial x} = -2\alpha x e^{-\alpha x^2} \tag{37}$$

and the second derivative is

$$\frac{\partial^2 \Psi_T}{\partial x^2} = 2\alpha(2\alpha x^2 - 1)e^{-\alpha x^2} \tag{38}$$

The expression for the local energy is then

$$E_L(x) = \frac{1}{\Psi_T(x)} H\Psi_T(x) = \alpha(1 - 2\alpha x^2) + \frac{1}{2}\omega^2 x^2 \tag{39}$$

In the three dimensional case we need the Laplacian of our trial wave function, and our single particle trial wave function is

$$\Psi_T(\mathbf{r}) = \exp[-\alpha(x^2 + y^2 + z^2)] \tag{40}$$

The Laplacian is then

$$\begin{aligned}\nabla^2\Psi_T &= 2\alpha(2\alpha x^2 - 1 + 2\alpha y^2 - 1 + 2\alpha z^2 - 1)\exp[-\alpha(x^2+y^2+z^2)] \\ &= 2\alpha(2\alpha r^2 - 3)e^{-\alpha r^2}\end{aligned} \tag{41}$$

and the local energy is

$$E_L(\mathbf{r}) = \frac{1}{\Psi_T(\mathbf{r})}H\Psi_T(\mathbf{r}) = \alpha(3 - 2\alpha r^2) + \frac{1}{2}\omega^2 r^2 \tag{42}$$

Similarly, for two dimensions we have

$$E_L(x,y) = \frac{1}{\Psi_T(x,y)}H\Psi_T(x,y) = \alpha[2 - 2\alpha(x^2+y^2)] + \frac{1}{2}\omega^2(x^2+y^2) \tag{43}$$

For N particles with the same mass the wave function becomes

$$\Psi_T(\mathbf{R}) = \prod_i e^{-\alpha r_i^2}, \tag{44}$$

while the Hamiltonian becomes

$$H = \sum_i^N \left(-\frac{1}{2}\nabla_i^2 + \frac{1}{2}\omega^2 r_i^2\right) \tag{45}$$

For particle k the three dimensional Laplacian of the wave function is then

$$\nabla_k^2\Psi_T(\mathbf{R}) = 2\alpha(2\alpha r_k^2 - 1)\prod_i e^{-\alpha r_i^2} = 2\alpha(2\alpha r_k^2 - 1)\Psi_T(\mathbf{R}) \tag{46}$$

The local energy for particle k in three dimensions is then

$$E_{L_k} = \alpha(3 - 2\alpha r_k^2) + \frac{1}{2}\omega^2 r_k^2 \tag{47}$$

and we find the total local energy by summing up the local energy of all N particles. The total local energy is then, in one, two and three dimensions respectively,

$$E_L = \sum_i^N \left(\alpha(1 - 2\alpha x_i^2) + \frac{1}{2}\omega^2 x_i^2\right) \tag{48}$$

$$E_L = \sum_i^N \left(\alpha[2 - 2\alpha(x_i^2 + y_i^2)] + \frac{1}{2}\omega^2(x_i^2 + y_i^2)\right) \tag{49}$$

$$E_L = \sum_i^N \left(\alpha(3 - 2\alpha r_i^2) + \frac{1}{2}\omega^2 r_i^2\right) \tag{50}$$

When we use importance sampling we also need a drift force given by

$$F = \frac{2\nabla\Psi_T}{\Psi_T}. \tag{51}$$

For the single particle wave function in one dimension the derivative is given by Eq.(37). In three dimensions the corresponding gradient of the wave function is

$$\nabla\Psi_T = -2\alpha e^{-\alpha r^2}\mathbf{r} \tag{52}$$

and the drift force of a single particle is then

$$F = -4\alpha\mathbf{r} \tag{53}$$

6.1.2 With Interaction

For the interacting case we need to use the complete trial wave function from Eq.(4). We rewrite the equation on an exponential form by defining $u(r_{ij}) = \ln f(r_{ij})$, with $r_{ij} = |\mathbf{r}_i - \mathbf{r}_j|$, so that

$$f(r_{ij}) = \exp\left(\sum_{i<j} u(r_{ij})\right). \tag{54}$$

We also use

$$g(\alpha, \beta, \mathbf{r}_i) = \exp\big(-\alpha(x_i^2 + y_i^2 + \beta z_i^2\big) = \phi(\mathbf{r}_i), \tag{55}$$

and the expression for the trial wave function then becomes

$$\Psi_T(\mathbf{R}) = \prod_i \phi(\mathbf{r}_i)\exp\left(\sum_{i<j} u(r_{ij})\right). \tag{56}$$

We need the gradient of this wave function for the drift force (for importance sampling), and we need the Laplacian to find a closed form expression for the local energy. We use the product rule to find the gradient. The gradient of the first factor is (using the product rule)

$$\begin{aligned}\nabla_k \prod_i \phi(\mathbf{r}_i) &= \prod_{i\neq k} \phi(\mathbf{r}_i)\nabla_k\phi(\mathbf{r}_k) \\ &= \nabla_k\phi(\mathbf{r}_k)\left[\prod_{i\neq k}\phi(\mathbf{r}_i)\right],\end{aligned} \tag{57}$$

since differentiating any factor in the product where $i \neq k$ results in zero. The gradient of the second factor in the trial wave function is

$$\begin{aligned}\nabla_k \exp\left(\sum_{i<j} u(r_{ij})\right) &= \exp\left(\sum_{i<j} u(r_{ij})\right)\nabla_k \sum_{i<j} u(r_{ij}) \\ &= \exp\left(\sum_{i<j} u(r_{ij})\right)\sum_{j\neq k}\nabla_k u(r_{kj}),\end{aligned} \tag{58}$$

since all the other terms in the second sum are zero when differentiating. To find $\nabla_k u(r_{kj})$ we use the chain rule. Each component of this gradient is then on the form

$$\begin{aligned}\frac{\partial u(r_{kj})}{\partial x_k} &= \frac{\partial r_{kj}}{\partial x_k}\frac{\partial u(r_{kj})}{\partial r_{kj}} \\ &= \frac{(x_k - x_j)}{r_{kj}}\frac{\partial u(r_{kj})}{\partial r_{kj}}.\end{aligned} \tag{59}$$

By defining

$$u'(r_{kj}) = \frac{\partial u(r_{kj})}{\partial r_{kj}}, \tag{60}$$

we then have

$$\nabla_k u(r_{kj}) = \frac{(\mathbf{r}_k - \mathbf{r}_j)}{r_{kj}}u'(r_{kj}). \tag{61}$$

From the product rule, the gradient of the wave function is then

$$\nabla_k \Psi_T(\mathbf{R}) = \nabla_k \phi(\mathbf{r}_k) \left[\prod_{i\neq k} \phi(\mathbf{r}_i)\right] \exp\left(\sum_{i<j} u(r_{ij})\right) + \prod_i \phi(\mathbf{r}_i) \exp\left(\sum_{i<j} u(r_{ij})\right) \sum_{j\neq k} \nabla_k u(r_{kj}) \tag{62}$$

For the closed form expression for the kinetic energy we need to find

$$\frac{1}{\Psi_T(\mathbf{R})} \nabla_k^2 \Psi_T(\mathbf{R}). \tag{63}$$

We differentiate the gradient and divide by $\Psi_T(\mathbf{R})$. For the first term in the gradient we get (again using the product rule)

$$\frac{\nabla_k^2 \phi(\mathbf{r}_k)}{\phi(\mathbf{r}_k)} + \frac{\nabla_k \phi(\mathbf{r}_k)}{\phi(\mathbf{r}_k)} \left(\sum_{j\neq k} \nabla_k u(r_{kj})\right) \tag{64}$$

For the second term of the gradient we get

$$\frac{\nabla_k \phi(\mathbf{r}_k)}{\phi(\mathbf{r}_k)} \left(\sum_{j\neq k} \nabla_k u(r_{kj})\right) + \sum_{j\neq k} \nabla_k u(r_{kj}) \sum_{j\neq k} \nabla_k u(r_{kj}) + \sum_{j\neq k} \nabla_k^2 u(r_{kj}), \tag{65}$$

where we can rewrite (if we want to)

$$\sum_{j\neq k} \nabla_k u(r_{kj}) \sum_{j\neq k} \nabla_k u(r_{kj}) = \sum_{i,j\neq k} \frac{(\mathbf{r}_k - \mathbf{r}_i)(\mathbf{r}_k - \mathbf{r}_j)}{r_{ki} r_{kj}} u'(r_{ki}) u'(r_{kj}). \tag{66}$$

We also have that

$$\nabla_k^2 u(r_{kj}) = \frac{(\mathbf{r}_k - \mathbf{r}_j)}{r_{kj}} \nabla_k u'(r_{kj}) + u'(r_{kj}) \nabla_k \frac{(\mathbf{r}_k - \mathbf{r}_j)}{r_{kj}}. \tag{67}$$

The first term is

$$\frac{(\mathbf{r}_k - \mathbf{r}_j)}{r_{kj}} \nabla_k u'(r_{kj}) = \frac{(\mathbf{r}_k - \mathbf{r}_j)}{r_{kj}} \frac{(\mathbf{r}_k - \mathbf{r}_j)}{r_{kj}} u''(r_{kj}) = u''(r_{kj}), \tag{68}$$

while the second term is

$$u'(r_{kj}) \nabla_k \frac{(\mathbf{r}_k - \mathbf{r}_j)}{r_{kj}} = \frac{2}{r_{kj}} u'(r_{kj}), \tag{69}$$

where we have used that

$$\nabla \frac{\mathbf{r}}{r} = \frac{2}{r}. \tag{70}$$

So we have

$$\nabla_k^2 u(r_{kj}) = u''(r_{kj}) + \frac{2}{r_{kj}} u'(r_{kj}). \tag{71}$$

The closed form expression for the kinetic energy is then

$$\begin{aligned}\frac{1}{\Psi_T(\mathbf{R})}\nabla_k^2\Psi_T(\mathbf{R}) &= \frac{\nabla_k^2\phi(\mathbf{r}_k)}{\phi(\mathbf{r}_k)} + 2\frac{\nabla_k\phi(\mathbf{r}_k)}{\phi(\mathbf{r}_k)}\sum_{j\neq k}\frac{(\mathbf{r}_k-\mathbf{r}_j)}{r_{kj}}u'(r_{kj}) \\ &+ \sum_{i,j\neq k}\frac{(\mathbf{r}_k-\mathbf{r}_i)(\mathbf{r}_k-\mathbf{r}_j)}{r_{ki}r_{kj}}u'(r_{ki})u'(r_{kj}) \\ &+ \sum_{j\neq k}\left(u''(r_{kj})+\frac{2}{r_{kj}}u'(r_{kj})\right),\end{aligned} \tag{72}$$

multiplied by $-\frac{1}{2}$. Similarly to what we did in the non-interacting case we find (with $\beta \neq 1$)

$$\frac{\nabla_k\phi(\mathbf{r}_k)}{\phi(\mathbf{r}_k)} = -2\alpha(x,y,\beta z), \tag{73}$$

$$\frac{\nabla_k^2\phi(\mathbf{r}_k)}{\phi(\mathbf{r}_k)} = 2\alpha(2\alpha x_k^2-1)+2\alpha(2\alpha y_k^2-1)+2\alpha(2\alpha\beta^2 z_k^2-\beta) \tag{74}$$

We also need the first and second derivative of $u(r_{kj}) = \ln f(r_{kj})$ when $r_{ij} > a$, in which case

$$u(r_{kj}) = \ln f(r_{kj}) = \ln\left(1-\frac{a}{r_{kj}}\right), \tag{75}$$

$$u'(r_{kj}) = \frac{1}{1-\frac{a}{r_{kj}}}\frac{a}{r_{kj}^2} = \frac{a}{r_{kj}(r_{kj}-a)}, \tag{76}$$

$$u''(r_{kj}) = \frac{a(a-2r_{kj})}{r_{kj}^2(r_{kj}-a)^2}. \tag{77}$$

We now have all we need to find the closed form solution for the kinetic energy in the interacting case.

6.2 Program Structure

The methods were implemented with a C++ program using object-orientation. The program consists of several classes responsible for different parts of the simulations. The classes are:

- **Hamiltonian:** A super-class for different Hamiltonians. The subclasses calculate the local energy for their specific Hamiltonian. Since calculating the kinetic energy with numerical differentiation is done the same for all Hamiltonians, this super-class is responsible for that. The subclasses are:
  - **HarmonicOscillator:** Calculates the local energy in the non-interacting case.
  - **HarmonicOscillatorRepulsive:** Calculates the local energy in the interacting case.
- **WaveFunction:** A super-class for different wave functions. The subclasses evaluate their specific wave function and also calculate the gradient, the Laplacian and the derivative w.r.t. the variational parameter α using the analytical expressions. The subclasses are:

  - **SimpleGaussian:** WaveFunction subclass for the non-interacting case.
  - **RepulsiveGaussian:** WaveFunction subclass for the interacting case.

- **InitialState:** A super-class for different initial states. The subclasses set up the initial state. The subclasses are:
  - **RandomUniform:** Sets up an initial state with uniformly distributed particle positions.

- **Particle:** Responsible for creating particles and adjusting their positions.

- **System:** Responsible for running the Monte Carlo simulation. It performs the Metropolis and Metropolis-Hastings algorithms.

- **Sampler:** Responsible for sampling interesting quantities and computing averages. It is also responsible for providing the data to the user, both by printing to terminal and saving to file.

- **SteepestDescent:** Responsible for optimizing variational parameters using the Steepest Descent method. Once the optimal parameter has been found, the System class is tasked with running a large Monte Carlo simulation.

- **Random:** Responsible for generating pseudo-random numbers according to different distributions.

There is also a main program which sets the necessary parameters and makes calls to the classes to start the simulation. An advantage to using an implementation like this is that it makes it easy to add functionality, like more wave functions, Hamiltonians and initial states, or alternative methods for optimizing variational parameters (e.g. the Conjugate Gradient method). The data analysis is done in Python, with the programs **blocking.py** and **density.py**.