FYS4411

Spring 2016

Variational Monte Carlo studies of bosonic systems

John-Anders Stende

Date: March 16, 2016

Abstract

The aim of this project is to use the Variational Monte Carlo (VMC) method to evaluate the ground state energy of a trapped, hard sphere Bose gas for different numbers of particles with a specific trial wave function.

Main findings

Contents

1 Introduction

Demonstrations of Bose-Einstein condensation (BEC) in gases of alkali atoms confined in magnetic traps has gained a lot of interest in the scientific community in recent years. Of interest is for instance the fraction of condensed atoms, the nature of the condensate and the excitations above the condensate.

An important feature of the trapped alkali systems is that they are dilute, i.e. the effective atom size is small compared to both the trap size and the inter-atomic spacing. In this situation the physics is dominated by two-body collisions, well discribed in terms of the s-wave scattering length a of the atoms. The condition for diluteness is defined by the gas parameter $x(\mathbf{r}) = n(\mathbf{r})a^3$, where $n(\mathbf{r})$ is the local density of the system. The theoretical framework of the Gross-Pitaevski equation is valid for $x_{av} \leq 10^{-3}$, but recent experiments have shown that the gas parameter may exceed this value due to the presence of so-called Feshbach resonance. Therefore, other methods like the VMC method may be needed.

In this project we evaluate the ground state energy of a trapped BEC by simulating different numbers of bosons in a harmonic oscillator potential in one, two and three dimensions. The energy is obtained using the VMC method, both with and without importance sampling. We have studied both the interacting and the non-interacting case, called system 1 and system 2 respectively. The method of blocking is utilized to do statistical analysis on the numerical data. We optimize the variational parameter α using the steepest descent method. The one-body density in the interacting and non-interacting case is also computed.

2 Theory

The trap we use is a spherical (S) or an elliptical (E) harmonic trap in one, two and three dimensions, with the latter 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} and ω_z defines the trap potential strength in the xy-plane and z-direction respectively. The two-body Hamiltonian is

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

and 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 so-called hard-core diameter of the bosons.

Our trial wave function for the ground state with N atoms is given by

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

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.,

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

The correlation wave function is

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

System 1 is non-interacting bosons ($a = 0$) in a spherical harmonic oscillator ($\beta = 1$). System 2 is interacting bosons ($a \geq 0$) in an elliptical harmonic oscillator ($\beta \neq 1$).

2.1 Benchmarks

We need to compare our results to exact theoretical closed-form expressions or other research to validate our code. N particles in a harmonic oscillator potential withouth interaction is a well known problem in quantum mechanics. The exact ground state energy and wavefunction for this system is (in natural units),

$$E = \frac{\omega d N}{2} \tag{7}$$

and

$$\Psi = \prod_{i=1}^N \exp\left(-\frac{1}{2} r_i^2\right) \tag{8}$$

where d is the number of spatial dimensions for each particle. Comparing (49) and (8), we see that $\alpha = 0.5$ yields $\Psi_T = \Psi$. Setting $\alpha = 0.5$ in our code should therefore reproduce the exact energies given by (7).

For system 2 there is no exact, analytical answer. Instead, we benchmark our results with [1] (for $d = 3$),

α	$N = 10$	$N = 50$	$N = 100$
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 for system 2

3 Methods

We use the *Variational Monte Carlo* (VMC) method in this project to obtain the ground state energy for our bosonic system. VMC applies the *variational principle* from quantum mechanics

$$E_0 \leq \frac{\langle \Psi_T | H | \Psi_T \rangle}{\langle \Psi_T | \Psi_T \rangle} \tag{9}$$

which states that the ground state energy is always less or equal than the expectation value of our Hamiltonian H for any trial wavefunction Ψ_T. VMC consists in choosing a trial wavefunction depending on one or more variational parameters, and finding the values of these parameters for which the expectation value of the energy is the lowest possible. The main challenge is to compute the multidimensional integral

$$\frac{\langle\Psi_T|H|\Psi_T\rangle}{\langle\Psi_T|\Psi_T\rangle} = \frac{\int d\mathbf{R}\Psi_T^*(\mathbf{R},\boldsymbol{\alpha})H(\mathbf{R})\Psi_T(\mathbf{R},\boldsymbol{\alpha})}{\int d\mathbf{R}\Psi_T^*(\mathbf{R},\boldsymbol{\alpha})\Psi_T(\mathbf{R},\boldsymbol{\alpha})} \tag{10}$$

where $\mathbf{R}$ is the positions of all the particles and $\boldsymbol{\alpha}$ is the set of variational parameters. Traditional integration methods like Gauss-Legendre methods are too computationally expensive, therefore other methods are needed.

3.1 Monte Carlo integration

Monte Carlo integration employs a non-deterministic approach to evaluate multidimensional integrals like (10), or in general

$$I = \int_\Omega f(\mathbf{x})d\mathbf{x} \tag{11}$$

Instead of using an explicit integration scheme, we sample points

$$\mathbf{x}_1 \ldots \mathbf{x}_N \in \Omega \tag{12}$$

according to some rule. The most naive approach is to use N uniform samples. The integral can then be approximated as the average of the function values at these points

$$I \approx \frac{1}{N}\sum_{i=1}^{N} f(\mathbf{x}_i) \tag{13}$$

This simple approach is however not very efficient, as it samples an equal amount of points in all regions of Ω, including those where f is zero.

3.2 Metropolis algorithm

A more clever approach is to sample points according to the probability distribution (PDF) defined by f. Such a PDF is in general difficult to obtain, thus we can't sample directly from it. Instead we use the Metropolis algorithm, which is a method to obtain random samples from a PDF for which direct sampling is difficult. These sample values are produced iteratively, with the distribution of the next sample being dependent only on the current sample value, thus making the sequence of samples into a Markov chain. We define $\mathbf{P}_i^{(n)}$ to be the probability for finding the system in state i at step n. The Metropolis algorithm is as follows:

- 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 recect the new state with probability $1 - A_{i\to j}$ and use state i as sample again

The transition probability T and the acceptance probability A must fulfill the principle of detailed balance

$$\frac{A_{i\to j}}{A_{j\to i}} = \frac{p_i T_{i\to j}}{p_j T_{j\to i}} \tag{14}$$

which ensures that $\mathbf{P}_i^{(n\to\infty)} \to p_i$, i.e. we end up at the correct distribution regardless of what we begin with.

The particles undergo a random walk under the guidance of the Metropolis algorithm. Defining the PDF

$$P(\mathbf{R}) = \frac{|\Psi_T(\mathbf{R})|^2}{\int |\Psi_T(\mathbf{R})|^2 d\mathbf{R}} \tag{15}$$

and the local energy (40), the integral (10) can be rewritten as

$$\langle E_L\rangle = \int P(\mathbf{R})E_L(\mathbf{R})d\mathbf{R} \tag{16}$$

and we see that our problem amounts to finding the expectation value of the local energy E_L on the PDF P. The VMC method approximates this integral as

$$\langle E_L\rangle \approx \frac{1}{N}\sum_{i=1}^{N} P(\mathbf{R}_i,\boldsymbol{\alpha})E_L(\mathbf{R}_i,\boldsymbol{\alpha}) \tag{17}$$

where N is the number of Monte Carlo cycles and $\mathbf{R}_i$ is the position of the particles at step i. The integral $\int |\Psi_T(\mathbf{R})|^2 d\mathbf{R}$ is very difficult to compute, but the Metropolis algorithm only needs a *ratio* of probabilities to decide if a move is accepted or not. This can be seen if we rewrite (14) as

$$\frac{p_j}{p_i} = \frac{T_{i\to j}A_{i\to j}}{T_{j\to i}A_{j\to i}} \tag{18}$$

In our case $p_j = P(\mathbf{R}_j)$ and $p_i = P(\mathbf{R}_i)$. The simplest form of the Metropolis algorithm, called brute force Metropolis, is to assume that the transition probability $T_{i\to j}$ is symmetric, implying that $T_{i\to j} = T_{j\to i}$; the ratio of probabilities (18) thus equals the ratio of acceptance probabilities. This leads to a description of the Metropolis algorithm where we accept or reject a new move by calculating the ratio

$$w = \frac{|\Psi_T(\mathbf{R}_j)|^2}{|\Psi_T(\mathbf{R}_i)|^2} \tag{19}$$

If $w \geq s$, where s is a random number $s \in [0, 1]$, the new position is accepted, else we stay at the same place. We now have the full machinery of the Monte Carlo approach to obtain the ground state energy of our bosonic system:

- Fix the number of Monte Carlo steps and choose the initial positions $\mathbf{R}$ and variational parameters $\boldsymbol{\alpha}$. Also set the step size $\Delta\mathbf{R}$ to be used when moving from $\mathbf{R}_i$ to $\mathbf{R}_j$.
- Initialize the local energy
- Choose a random particle
- Calculate a trial position $\mathbf{R}_j = \mathbf{R}_i + r\dot{\Delta}\mathbf{R}$ where r is a random variable $r \in [0, 1]$
- Use the Metropolis algorithm to accept or reject this move by calculating the ratio (19). If $w \geq s$, where s is a random number $s \in [0, 1]$, the new position is accepted, else we stay at the same place.
- If the step is accepted, set $\mathbf{R} = \mathbf{R}_j$ for the chosen particle
- Update the local energy

When the Monte Carlo sampling is finished, we calculate the mean local energy, which is our approximation of the ground state energy of the system. The Metropolis algorithm is implemented as follows:

Listing 1: Brute Forde Metropolis algorithm

```
int particle = Random::nextInt(m_numberOfParticles);     // choose random particle
int dimension = Random::nextInt(m_numberOfDimensions);   // choose random dimension
double change = (Random::nextDouble()*2-1)*m_stepLength;   // propose change

// get old wavefunction
double waveFuncOld = m_waveFunction->evaluate(m_particles);

// adjust position
m_particles[particle]->adjustPosition(change, dimension);

// get new wavefunction
double waveFuncNew = m_waveFunction->evaluate(m_particles);

// accept/reject new position using Metropolis algorithm
double ratio = pow(waveFuncNew, 2) / pow(waveFuncOld, 2);

if (ratio >= Random::nextDouble()) {
    //cout << m_particles[particle]->getPosition()[0] << endl;
    return true;
}
else {
    // correct position change
    m_particles[particle]->adjustPosition(-change, dimension);
    return false;
}
```

3.3 Importance sampling

A more efficient way to do Monte Carlo sampling is to replace the brute force Metropolis algorithm with a walk in coordinate space biased by the trial wavefunction. This approach is based on the Fokker-Planck equation and the Langevin equation for generating a trajectory in coordinate space.

The Langevin equation is a stochastic differential equation

$$\frac{\partial x(t)}{\partial t} = DF(x(t)) + \eta \tag{20}$$

where D is the diffusion constant and η a random variable. The new positions y in coordinate space are the solutions of (20) using Euler's method:

$$y = x + DF(x)\Delta t + \xi\sqrt{\Delta t} \tag{21}$$

where ξ is a gaussian random variable and Δt is a chosen time step. D is equal to $1/2$ which comes from the factor $1/2$ in the kinetic energy operator. Δt is to be viewed as a parameter which yields stable values of the ground state energy for values $\Delta t \in [0.001, 0.01]$. (21) is similar to the brute force Metropolis equation for updating positions except for the term containing $F(x)$. This is the function that pushes the particles towards regions of configuration space where the wavefunction is large, in contrast to the brute force method where all regions are equally probable. In three dimension $F(x)$ is called the *drift vector* $\mathbf{F}(\mathbf{x})$. The Monte Carlo method with the Metropolis algorithm can be seen as isotropic diffusion process by a time-dependent probability density, with or without a drift. corresponding to brute force and importance samling respectively. The drift vector can be found from the Fokker-Planck equation

$$\frac{\partial P}{\partial t} = \sum_i D\frac{\partial}{\partial \mathbf{x}_i}\left(\mathbf{x}_i - \mathbf{F}_i\right)P(\mathbf{x}, t) \tag{22}$$

The convergence to a stationary probability density can be obtained by setting the left hand side to zero. The resulting equation is only satisfied if all terms of the sum are equal to zero,

$$\frac{\partial^2 P}{\partial \mathbf{x}_i^2} = P\frac{\partial}{\partial \mathbf{x}_i}\mathbf{F}_i + \mathbf{F}_i\frac{\partial}{\partial \mathbf{x}_i}P \tag{23}$$

The drift vector should have the form $\mathbf{F} = g(\mathbf{x})\frac{\partial P}{\partial \mathbf{x}}$. Inserting this in (23) yields

$$\mathbf{F} = 2\frac{1}{\Psi_T}\nabla\Psi_T \tag{24}$$

which is known as the *quantum force*.

The Monte Carlo method with the Metropolis algorithm can be seen as isotropic diffusion process by a time-dependent probability density, with or without a drift. corresponding to brute force and importance samling respectively. Our new transition probabilty is thus a solution to the Fokker-Planck equation, yielding

$$G(y, x, \Delta t) = \frac{1}{(4\pi D\Delta t)^{3N/2}}\exp(-(y - x - D\Delta t F(x))^2/4D\Delta t) \tag{25}$$

which in turn means that our brute force Metropolis accept/reject ratio (19) is replaced by the so-called Metropolis-Hastings article

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

The Metropolis Hastings algorithm has the same role as the brute force method in the Monte Carlo approach to obatin expectation values, now with (19) replaced by (26) and trial position calculated according to (21)

3.4 Steepest descent method

We turn now to the problem of finding the variational parameters that minimizes the expectation value of the local energy $\langle E_L(\mathbf{R}, \alpha)\rangle$. This project considers a trial wavefunction with only one variational parameter α. There are many optimization algorithms to choose from, we have chosen the Steepest descent method due to its simplicity. This method finds a local minimum of a function by taking steps proportional to the negative gradient of the function at a given point, i.e. where the function has the steepest descent. The algorithm is as follows:

- Choose an initial α_0 and step length γ_0.
- For $i \geq 0$: Compute $\alpha_{i+1} = \alpha_i - \gamma_i\frac{d\langle E_L(\mathbf{R},\alpha)\rangle}{d\alpha}$

- Continue until a maximum number of steps are performed or $|\frac{d\langle E_L(\mathbf{R},\alpha)\rangle}{d\alpha}|$ is less than some tolerance

We should get $\langle E_L(\mathbf{R},\alpha_i)\rangle \geq \langle E_L(\mathbf{R},\alpha_{i+1})\rangle \geq \ldots$. If this is not the case, we reject the new step and decrease the step size by a factor of 2 to obtain a more accurate value. $\langle E_L(\mathbf{R},\alpha)\rangle$ is as we have seen a multidimensional integral, and the derivative w.r.t. α is not easily computed. Let us define

$$\bar{E}_\alpha = \frac{d\langle E_L(\alpha)\rangle}{d\alpha} \tag{27}$$

and

$$\bar{\Psi}_T = \frac{d\Psi_T(\alpha)}{d\alpha} \tag{28}$$

Using the chain rule and the hermicity of the Hamiltonian it can be shown that

$$\bar{E}_\alpha = 2\left(\Big\langle \frac{\bar{\Psi}_T}{\Psi_T(\alpha)} E_L(\alpha)\Big\rangle - \Big\langle \frac{\bar{\Psi}_T}{\Psi_T(\alpha)}\Big\rangle \langle E_L(\alpha)\rangle\right) \tag{29}$$

thus we need the expectation values of

$$\frac{\bar{\Psi}_T}{\Psi_T(\alpha)} E_L(\alpha) \tag{30}$$

and

$$\frac{\bar{\Psi}_T}{\Psi_T(\alpha)} \tag{31}$$

The complete VMC method then amounts to the following:

- Make initial guess α_0
- Run 10^4-10^5 Metropolis steps, sample (30) and (31)
- Compute (29)
- Calculate new α using the Steepest descent method

The above steps are repeated until the value of α is sufficiently accurate, before a new round of Metropolis steps are run, this time with many cycles (10^6-10^8). We then obtain our approximation for the ground state energy of the system.

The Steepest descent method is implemented as follows:

Listing 2: The Steepest Descent method

```
void SteepestDescent::optimize(double initialAlpha) {

    int maxNumberOfSteps = 30;
    double tolerance = 0.001;
    double oldAlpha = initialAlpha;
    for (int i=0; i < maxNumberOfSteps; i++) {

        // make initial state
        m_system->getInitialState()->setupInitialState();

        // set value of alpha
        m_system->getWaveFunction()->setAlpha(oldAlpha);

        // run metropolis steps
        m_system->runMetropolisSteps((int) 1e4, false , false , false);

        // compute derivative of exp. value of local energy w.r.t. alpha
        double localEnergyDerivative = 2 *
                             ( m_system->getSampler()->getWaveFunctionEnergy() -
                               m_system->getSampler()->getWaveFunctionDerivative() *
                               m_system->getSampler()->getEnergy()  );

        cout << "localEnergyDerivative = " << localEnergyDerivative << endl;
        // compute new alpha
        double newAlpha = oldAlpha - m_stepLengthOptimize*localEnergyDerivative;
        cout << "newAlhpa = " << newAlpha << endl;
        cout << "oldAlpha = " << oldAlpha << endl;
        cout << std::abs(newAlpha-oldAlpha) << endl;
        if ( std::abs(newAlpha - oldAlpha) < tolerance ) {
            break;
        }
        // before new iteration
        oldAlpha = newAlpha;
    }
    cout << "Optimal alpha = " << oldAlpha << endl;

    // run many Metropolis steps with the optimal alpha

    // make initial state
    m_system->getInitialState()->setupInitialState();

    // set value of alpha
    m_system->getWaveFunction()->setAlpha(oldAlpha);

    // run metropolis steps
    m_system->runMetropolisSteps((int) 1e7, false, false, false);
}
```

.

3.5 Blocking

Monte Carlo simulations can be treated as computer experiments. The results can be analyzed with the same statistical tools as we would use analyzing experimental data. We are looking for expectation values of these data, and how accurate they are. A stochastic process like a Monte Carlo experiment produces sequentially a chain of values

$$\{x_1, x_2 \dots x_k \dots x_n\} \tag{32}$$

called a sample. Each value x_k is called a measurement. The sample variance

$$\mathrm{var}(x) = \frac{1}{n}\sum_{k=1}^{n}(x_k - \bar{x}_n) \tag{33}$$

where $\bar{x}_n$ is the sample mean, is a measure of the statistical error of a *uncorrelated* sample. However, a Monte Carlo simulation produces a correlated sample, thus we need another measure of the sample error. The sample covariance

$$\mathrm{cov}(x) \equiv \frac{1}{n}\sum_{kl}(x_k - \bar{x}_n)(x_l - \bar{x}_n) \tag{34}$$

is a measure of the sequential correlation between succeding measurements of a sample. (Note that these are experimental values for the sample, not the *true* properties of the stochastic variables, which we need an infinite number of measurements to calculate).

It can be shown that an estimate of the error err_X of a correlated sample is

$$\mathrm{err}_X = \frac{1}{n}\mathrm{cov}(x) \tag{35}$$

With the help of the *autocorrelation function* from statistical theory we can rewrite this error as

$$\mathrm{err}_X = \frac{\tau}{n}\mathrm{var}(x) \tag{36}$$

where τ is the *autocorrelation time* which accounts for the correlation between measurements. In the presence of correlation the effective number of measurements becomes

$$n_{\mathrm{eff}} = \frac{n}{\tau} \tag{37}$$

Neglecting τ thus gives an error estimate that is less than the true sample error. The autocorrelation time is however expensive to compute. We can avoid the computation of this quantity by using the technique of blocking. The idea behind this method is to split the sample into blocks, find the mean of each block and then calculate the total mean and variance of all the block means. This is done for increasing block sizes n_b until the measurements of two sequential blocks are uncorrelated, enabling us to extract the value of $\tau = n_b \Delta t$. The true sample error,

$$\sigma = \left(\frac{1 + 2\tau/\Delta t}{n} \left(\langle E_L^2 \rangle - \langle E_L \rangle^2 \right) \right)^{1/2} \tag{38}$$

can then be calculated.

The blocking algorithm is as follows:

- Do a Monte Carlo simulation, store the local energy for each step to file
- Read the file into an array
- Loop over increasing block sizes:
  - For each block size n_b, loop over array in steps of n_b taking the mean of elements $[in_b, (i+1)n_b], \dots$
  - Calculate total mean and variance of all block means and store
- Plot total variance for all block sizes.
- Extract τ and compute (38)

3.6 Implementation

We have made an object-oriented code in C++. We give here an overview of the class structure and what the different classes do,

- *Main program*: Sets all the parameters needed to a simulation.
- *System*: Runs the Monte Carlo cycles with/without importance sampling
- *Sampler*: Samples quantities we want to measure for each cycle and computes expectation values
- *Particle*: Sets and adjusts particle positions
- *SteepestDescent*: Runs the steepest descent method
- *InitialState*: Super-class for setting up different initial states
  - *RandomUniform*: Assigns initial positions according to a uniform distribution
- *WaveFunction*: Super-class for different wave functions. Sets the variational parameters. All subclasses must implement functions to evalute the analytical expressions for Ψ_T, $\nabla\Psi_T$, $\nabla^2\Psi_T$ and $d\Psi_T/d\alpha$.
  - *SimpleGaussian*: Implements the above quantities for Ψ_T used in system 1
  - *InteractingGaussian*: Implements the above quantities for Ψ_T used in system 2
- *Hamiltonian*: Super-class for different Hamiltonians. Calculates E_L by computing potential and kinetic energy, either numerically or analytically for the latter. The analytical Laplacian is obtained from *WaveFunction.*
  - *HarmonicOscillator*: Hamiltonian for system 1
  - *HarmonicOscillatorInteracting*: Hamiltonian for system 2

In addition to these we use class *Random* to generate pseudo-random numbers.

All the information are stored in System in the form of class objects of the other classes, which in turn recieves the System object so that they can access this information via setters and getters in the System class. This way of communicating between classes limits the user's capability to alter vital functionality and also makes the program more user-friendly. Object-oriented code is also easy to expand on. We don't need to add new functionality to e.g. implement a new wave function; only the specifics of this new wave function needs to be implemented.

4 Results

We investigate two different systems, both with only one variational parameter α:

1. $N = 1$, $N = 10$, $N = 100$ and $N = 500$ bosons in one, two and three dimensions in a spherical harmonic oscillator potential with no interaction.
2. $N = 10$, $N = 50$ and $N = 100$ bosons in three dimensions in an elliptical harmonic oscillator potential including interaction and a correlated trial wavefunction.

We calculate the ground state energy for system 1 both with and without importance sampling. Only the brute force Metropolis algorithm is applied for system 2. The kinetic energy is calculated both numerically and analytically for system 1, referred to as the *numerical method* and the *analytical method* respectively. Only the analytical method is applied for system 2.
We use blocking to analyze the error of the statistical data for both systems. The steepest descent method is applied to optimize α. In addition to this, the one-body density is computed for both systems.

4.1 System 1 - brute force Metropolis

The parameters used to produce the below results are as follows

Listing 3: Parameters brute force Metropolis system 1

```
    int numberOfSteps       = (int) 1e4;
    double omega            = 1.0;        // oscillator frequency
    double alpha            = 0.5;        // variational parameter 1
    double stepLength       = 1.5;        // metropolis step length
    double equilibration    = 0.1;
```

We want to valide our code using the benchmark (7) and compare the CPU time difference for computing kinetic energy numerically vs analytically. The Metropolis step length is set so that the acceptance rate equals about 0.5. The number of Metropolis steps is only $1e4$ because setting $\alpha = 0.5$ is equivalent to having a trial wavefunction that is exact for this system, according to (8). Thus, a few steps should suffice to obtain the exact energy.

N	d	⟨E⟩	σ	CPU time
1	1	0.5	0	5.53e-3
1	2	1	0	5.66e-3
1	3	1.5	0	5.70e-3
10	1	5	0	7.20e-3
10	2	10	0	7.36e-3
10	3	15	0	1.00-3
100	1	50	0	0.0215
100	2	100	0	0.223
100	3	150	0	0.274
500	1	250	0	0.0843
500	2	500	0	0.110
500	3	750	0	0.131

Table 2: Analytical kinetic energy

N	d	$\langle E_L \rangle$	σ	CPU time
1	1	0.5	9.87e-8	6.96e-3
1	2	1	1.873-7	0.0151
1	3	1.5	1.88e-7	0.0157
10	1	5	4.42e-7	0.0314
10	2	10	6.78e-7	0.0598
10	3	15	1.11e-6	0.0955
100	1	50	2.78e-6	0.743
100	2	100	6.18e-6	1.78
100	3	150	8.95e-6	3.16
500	1	200	1.32e-5	15.1
500	2	500	2.52e-5	41.5
500	3	750	6.65e-5	74.5

Table 3: Numerical kinetic energy

As expected, the CPU time increases with N and d (number of dimensions). Calcuating the kinetic energy analytically speeds up the simulation a great deal and the CPU time difference increases exponentially with N.
The analytical approach reproduces the exact energies with zero error. Calculating the kinetic energy numerically results in a small deviation due to the inherent error of the differentiation algorithm that increases with N.

4.2 System 1 - importance sampling

With the following parameters,

Listing 4: Parameters importance sampling Metropolis system 1

```
    int numberOfSteps       = (int) 1e4;
    double omega            = 1.0;          // oscillator frequency
    double alpha            = 0.5;          // variational parameter 1
    double stepLength       = 1.5;          // metropolis step length
    double equilibration    = 0.1;          // amount of the total steps
    double timeStep         = 0.001;        // importance sampling
```

we obtain

N	d	$\langle E_L \rangle$	σ	Time
1	1	0.5	0	0.0183
1	2	1	0	0.0338
1	3	1.5	0	0.0501
10	1	5	0	0.0479
10	2	10	0	0.0488
10	3	15	0	0.0812
100	1	50	0	0.0605
100	2	100	0	0.1212
100	3	150	0	0.218
500	1	200	0	0.171
500	2	500	0	0.451
500	3	750	0	0.941

Table 4: Analytical

N	D	$\langle E_L \rangle$	σ	CPU time
1	1	0.5	9.41e-8	0.0170
1	2	1	1.48e-7	0.0352
1	3	1.5	1.58e-7	0.0618
10	1	5	3.42e-7	0.0413
10	2	10	3.50e-7	0.104
10	3	15	9.23e-7	0.163
100	1	50	3.02e-6	0.737
100	2	100	5.72e-6	1.889
100	3	150	6.88e-6	3.319
500	1	200	1.01e-5	15.509
500	2	500	4.94e-5	41.047
500	3	750	2.42e-5	75.584

Table 5: Numerical

There are two differences between the brute force algorithm and Metropolis with importance sampling. Firstly, the CPU time increases because we have to calculate the drift vector and Green's function for each Monte Carlo cycle. Secondly, the error (for numerical kinetic energy) *decreases*, as we should expect considering the more efficient way of sampling positions compared to brute force sampling.

We also want to investigate how the results depend on the chosen time step. For $N = 100$ and $d = 3$ we get,

Time step	Acceptance rate	σ
0.001	0.871	0
0.005	0.831	0
0.01	0.801	0

Table 6: Analytical

Time step	Acceptance rate	σ
0.001	0.871	6.88e-6
0.005	0.831	7.39e-6
0.01	0.801	9.91e-6

Table 7: Numerical

We see that the acceptance rate increases while the error decreases for increasing time step. This make sense because accepting fewer steps leads to fewer energy samples??? This reduces the statistical error, but there souldn't be statistical errors in this case????

4.3 System 2

Before running the simulation for system 2, we must optimize the variational parameter α using the steepest descent method. We run $1e5$ number of Metropolis steps for each iteration. The set of parameters

Listing 5: Parameters system 2

```
double omega              = 1.0;         // oscillator frequency
double beta               = 2.82843;     // variational parameter 2
double stepLength         = 1.5;         // metropolis step length
double equilibration      = 0.1;         // amount of the total steps
double a                  = 0.0043;      // hard sphere radius
double gamma              = 2.82843;     // trap potential strength z-direction

double initialAlpha = 0.7;
double stepLengthOptimize = 0.01;
```

yields the the optimal variational parameter $\alpha = 0.500605$. We now run simulations for $N = 10$, $N = 50$ and $N = 100$ bosons with this optimal α. The number of Monte Carlo steps is $1e6$ for $N = 10$ and $1e5$ for $N = 50$ and $N = 100$.

N	$\langle E_L \rangle$	σ	CPU time
10	24.143	7.14e-3	9.027
50	120.51	0.121	87.86
100	239.5	1.114	687.66

Table 8: Numerical

These energies are quite close to those in 1 from [1]. Reproducing these excactly is not expected, as our set of parameters are not equal to those used in [1]. The energy values are not that different from those of system 1, especially for $N = 10$ and $N = 50$. This tells us that the correlation effects are not very dominant, due to the dilute nature of our system. A denser system would have had larger correlations.

4.3.1 Error analysis

As discussed above, our estimate of error for system 2 is too low because the correlation effects are not included in the error estimate. A proper error analysis is done using the blocking method. First, we plot the standard deviation σ as a function of block size n_b,

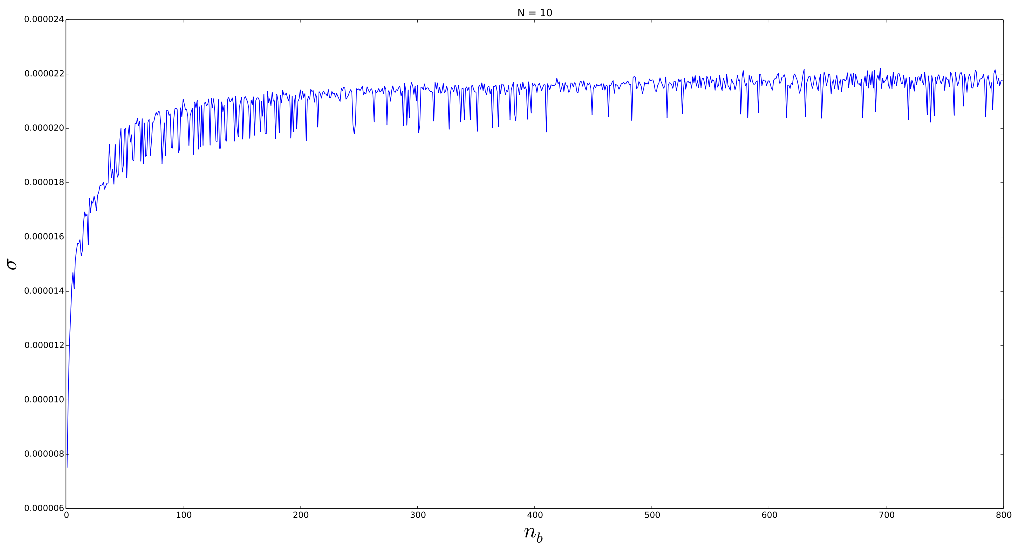

Figure 1: N = 10

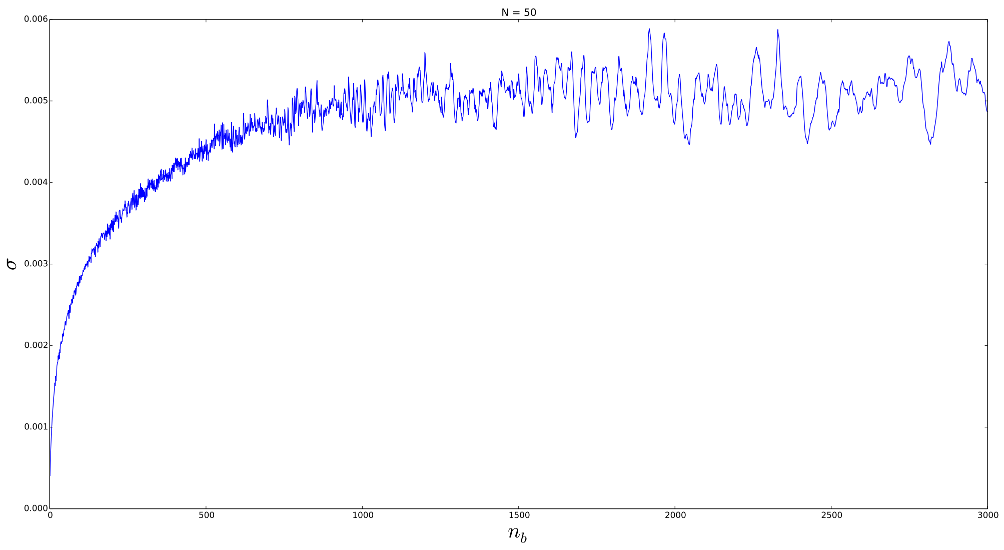

Figure 2: N = 50

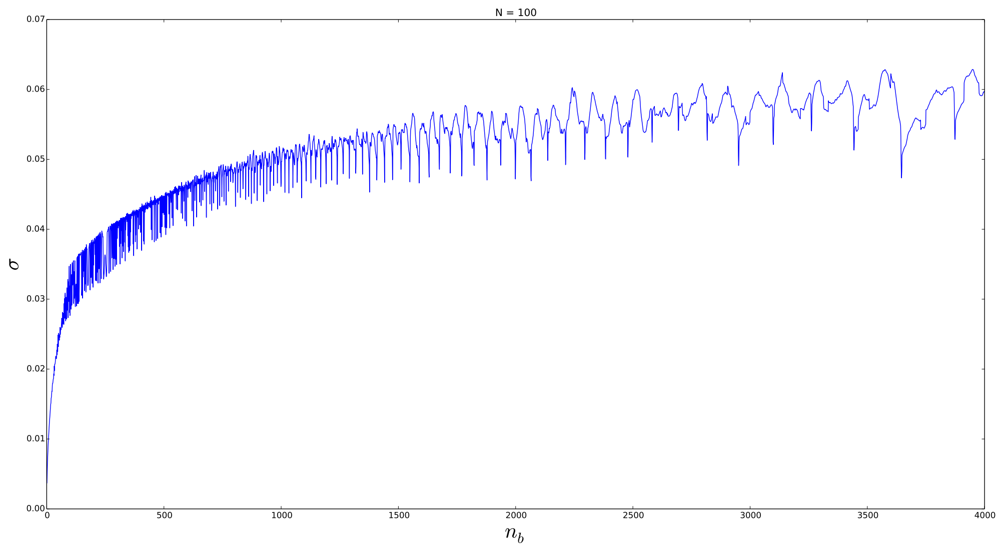

Figure 3: N = 100

We can now estimate τ by looking at which block size n_b the plateau is reached and multiply this value with the above Monte Carlo step size,

- N=10: $n_b \approx 100 \Rightarrow \tau = 100 \cdot 1.5 = 150$
- N=50: $n_b \approx 1000 \Rightarrow \tau = 1000 \cdot 1.5 = 1500$
- N=100: $n_b \approx 1500 \Rightarrow \tau = 1500 \cdot 1.5 = 2250$

The true sample errors are then, according to (38),

- N=10: $\sigma_{true} = 0.032$
- N=50: $\sigma_{true} = 0.77$
- N=100: $\sigma_{true} = 6.1$

4.4 One-body density

The one-body density is defined as

$$\rho(\mathbf{r}) = \int d\mathbf{r}_2 \dots d\mathbf{r}_N |\Psi(\mathbf{r}, \mathbf{r}_2, \dots, \mathbf{r}_N)|^2 \tag{39}$$

This quantity can be visuzalized by making a histogram of the radial distance of the bosons. The following paramters,

Listing 6: Parameters system 2

```
    int numberOfDimensions  = 3;
    int numberOfParticles   = 30;
    int numberOfSteps       = (int) 1e6;
    double omega            = 1.0;          // oscillator frequency
    double alpha            = 0.5;          // variational parameter 1
    double beta             = 2.82843;      // variational parameter 2
    double stepLength       = 1.5;          // metropolis step length
    double equilibration    = 0.1;          // amount of the total steps used
    double a                = 0.0043;       // hard sphere radius
    double gamma            = 2.82843;      // trap potential strength z-direction
```

results in

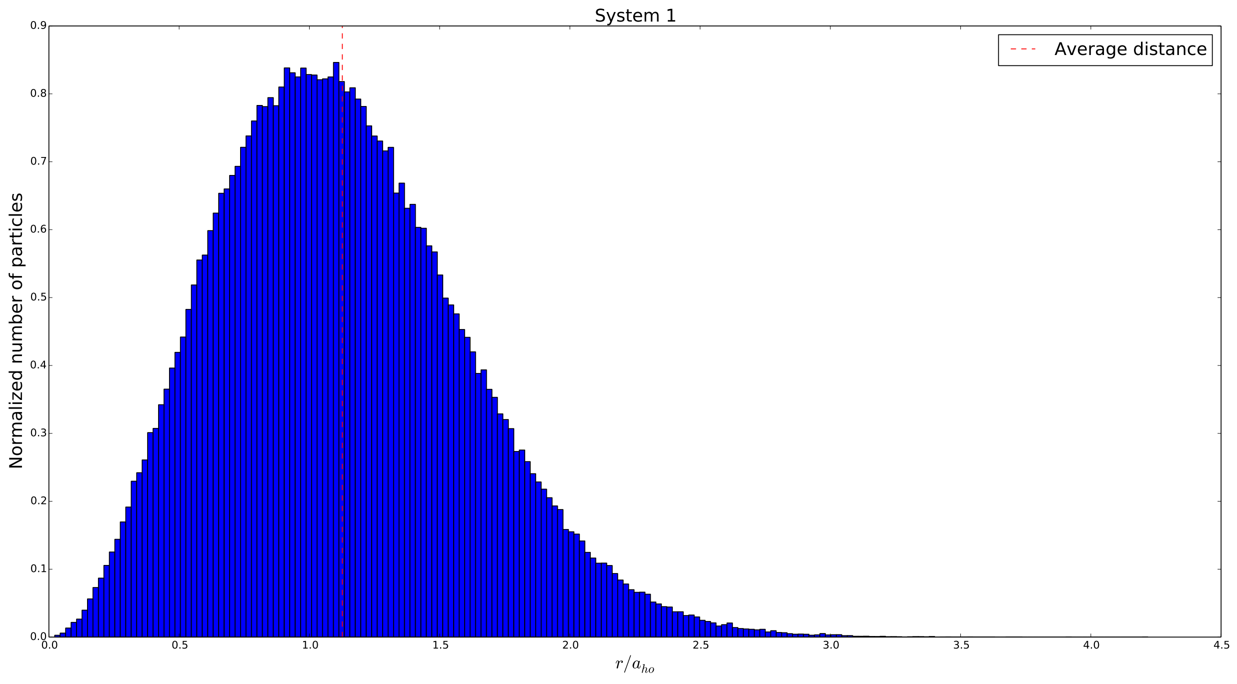

Figure 4: Radial distribution of particles - system 1

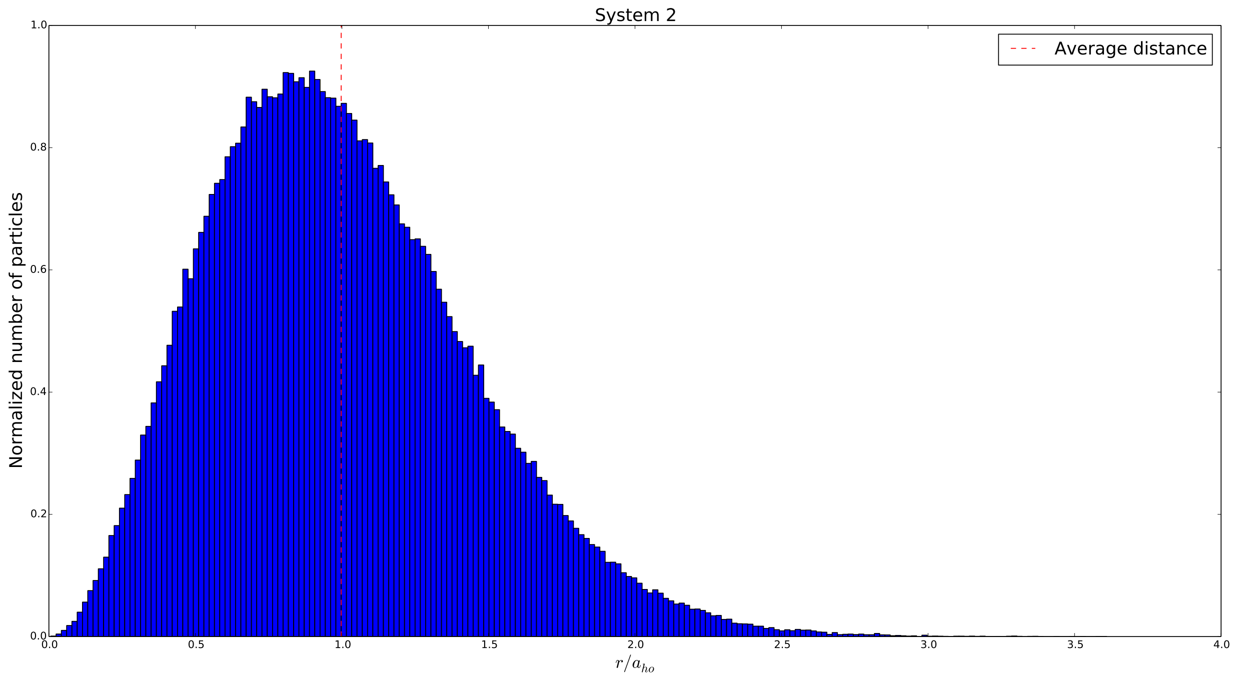

Figure 5: Radial distribution of particles - system 2

The distribution for system 1 is shifted towards the right compared to system 2, thus the average radial distance is larger.

The distribution of particles in the xy-plane is included for visualization purposes.

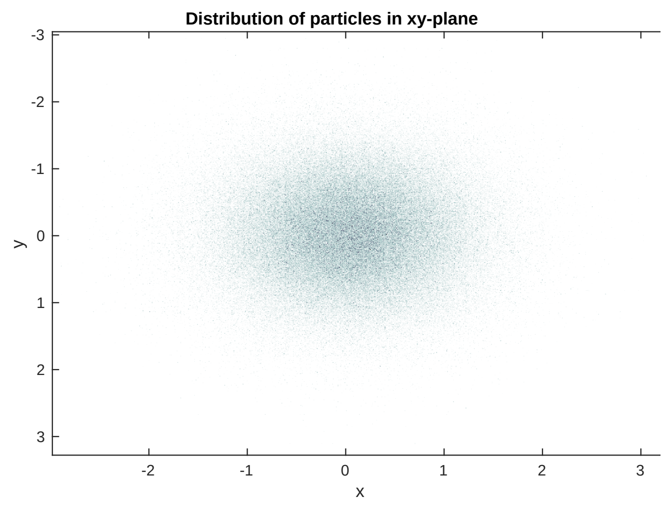

Figure 6: System 1

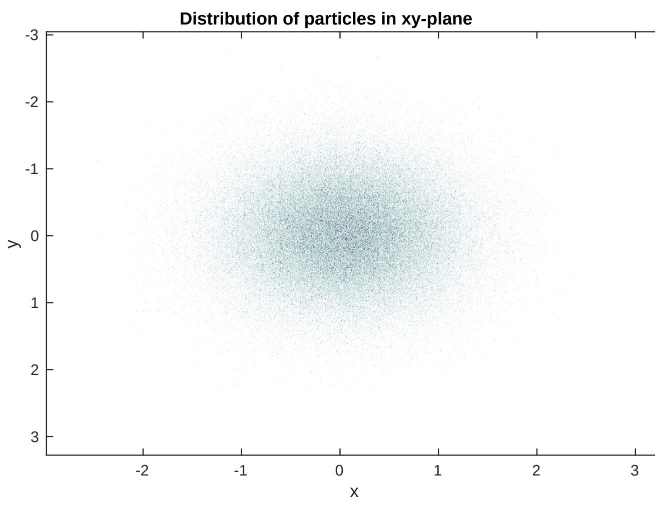

Figure 7: System 2

5 Conclusions

6 Appendix

6.1 Closed-form expressions

The quantity we are aiming to compute is the expectiation value of the so-called local energy

$$E_L(\mathbf{R}) = \frac{1}{\Psi_T(\mathbf{R})} H\Psi_T(\mathbf{R}), \tag{40}$$

We can find closed-form expressions for the local energy with our specific Hamiltonian H and trial wavefunction Ψ_T. Computing the local energy involves a second derivative of Ψ_T, which can be expensive to compute numerically. Analytical expressions are therefore useful, as they can speed up the computations.

First, we find the local energy with only the (spherical) harmonic oscillator potential, that is we set $a = 0$ and $\beta = 1$. During these calcuations we will use natural units, thus $\hbar = m = 1$. For one particle in one dimension we have

$$\Psi_T(x) = e^{-\alpha x^2} \tag{41}$$

and

$$H = -\frac{1}{2}\frac{\partial^2}{\partial x^2} + \frac{1}{2}\omega x^2 \tag{42}$$

The second derivate of the trial wave function is

$$\frac{\partial^2 \Psi_T}{\partial x^2} = 2\alpha e^{-\alpha x^2}(2\alpha x^2 - 1) \tag{43}$$

so that

$$E_L = \frac{1}{\Psi_T} H\Psi_T = \alpha(1 - 2\alpha x^2) + \frac{1}{2}\omega^2 x^2 \tag{44}$$

In three dimensions the double derivative is replaced by the Laplacian when computing the kinetic energy

$$\frac{1}{\Psi_T} \bigtriangledown^2 \Psi_T = 2\alpha(2\alpha x^2 - 1) + 2\alpha(2\alpha y^2 - 1) + 2\alpha(2\alpha z^2 - 1) \tag{45}$$

$$= 2\alpha(2\alpha r^2 - 3) \tag{46}$$

thus the local energy is

$$E_L = \frac{1}{\Psi_T}\left(-\frac{1}{2} \bigtriangledown^2 \Psi_T + V_{ext}\right) = \alpha(3 - 2\alpha r^2) + \frac{1}{2}\omega^2 r^2 \tag{47}$$

We now turn our attention to N particles, with the following wavefunction and Hamiltonian

$$\Psi_T(\mathbf{R}) = \prod_i e^{-\alpha r_i^2} \tag{48}$$

$$H = \sum_i^N \left(-\frac{1}{2}\bigtriangledown_i^2 + \frac{1}{2}\omega^2 r_i^2\right) \tag{49}$$

The first term of the k-th Laplacian of this wavefunction is

$$\nabla_k^2 \prod_i e^{-\alpha r_i^2} = 2\alpha(2\alpha x_k^2 - 1) \prod_i e^{-\alpha r_i^2} \tag{50}$$

and when we divide with Ψ_T to obtain the the local energy we end up with

$$E_L = \sum_i^N \left(\alpha(3 - 2\alpha r_i^2) + \frac{1}{2}\omega^2 r_i^2 \right) \tag{51}$$

For one dimension the expression is

$$E_L = \sum_i^N \left(\alpha(1 - 2\alpha x_i^2) + \frac{1}{2}\omega^2 x_i^2 \right) \tag{52}$$

It is also useful to compute the analytical expression for the drift force F to be used in importance sampling

$$F = \frac{2\nabla\Psi_T}{\Psi_T}. \tag{53}$$

The gradient of Ψ_T is

$$\begin{aligned} \nabla\Psi_T &= (-2\alpha x, -2\alpha y, -2\alpha z)e^{-\alpha r^2} & (54) \\ &= -2\alpha e^{-\alpha r^2}\mathbf{r} & (55) \end{aligned}$$

where $\mathbf{r} = (x, y, z)$. Dividing by the wavefunction and multiplying with 2

$$F = -4\alpha\mathbf{r} \tag{56}$$

6.1.1 Analytic energy for interacting system

We rewrite (4),

$$\Psi_T(\alpha, \beta, \mathbf{R}) = \prod_i g(\alpha, \beta, \mathbf{r}_i)\text{exp}\left(\sum_{i<j} u(r_{ij}) \right) \tag{57}$$

where $r_{ij} = |\mathbf{r}_i - \mathbf{r}_j|$ and $u(r_{ij}) = \ln f(r_{ij})$. We also set

$$g(\alpha, \beta, \mathbf{r}_i) = \exp\big(- \alpha(x_i^2 + y_i^2 + \beta z_i^2\big) = \phi(\mathbf{r}_i) \tag{58}$$

so that our trial wave function can be written

$$\Psi_T(\mathbf{R}) = \prod_i \phi(\mathbf{r}_i)\text{exp}\left(\sum_{i<j} u(r_{ij}) \right) \tag{59}$$

We need the Laplacian of this wave function to compute kinetic energy analytically. We start with finding the gradient,

$$\begin{aligned} \nabla_k \prod_i \phi(\mathbf{r}_i) &= \prod_{i\neq k} \phi(\mathbf{r}_i)\nabla_k\phi(\mathbf{r}_k) & (60) \\ &= \nabla_k\phi(\mathbf{r}_k) \prod_{i\neq k} \phi(\mathbf{r}_i) & (61) \end{aligned}$$

and

$$\begin{aligned} \nabla_k\text{exp}\left(\sum_{i<j} u(r_{ij}) \right) &= \text{exp}\left(\sum_{i<j} u(r_{ij}) \right) \nabla_k \sum_{i<j} u(r_{ij}) & (62) \\ &= \text{exp}\left(\sum_{i<j} u(r_{ij}) \right) \sum_{j\neq k} \nabla_k u(r_{kj}) & (63) \end{aligned}$$

where the last equality follows because all terms that lack k as index in the sum $\sum_{i<j} u(r_{ij})$ will diseappear when differentiating. The first compononet of the k-th gradient of $u(r_{kj})$ is

$$\frac{\partial u(r_{kj})}{\partial x_k} = \frac{\partial r_{kj}}{\partial x_k}\frac{\partial u(r_{kj})}{\partial r_{kj}} \tag{64}$$

$$= \frac{(x_k - x_j)}{r_{kj}}u'(r_{kj}) \tag{65}$$

where we have defined

$$u'(r_{kj}) = \frac{\partial u(r_{kj})}{\partial r_{kj}} \tag{66}$$

thus

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

Using the product rule, the total gradient equals

$$\nabla_k \Psi_T(\mathbf{R}) = \nabla_k \phi(\mathbf{r}_k)\left[\prod_{i\neq k}\phi(\mathbf{r}_i)\right]\exp\left(\sum_{i<j}u(r_{ij})\right) + \prod_i \phi(\mathbf{r}_i)\exp\left(\sum_{i<j}u(r_{ij})\right)\sum_{j\neq k}\frac{(\mathbf{r}_k - \mathbf{r}_j)}{r_{kj}}u'(r_{kj}) \tag{68}$$

Differentiating the first term of this expression and dividing with $\Psi_T(\mathbf{R})$,

$$\frac{\nabla_k^2 \phi(\mathbf{r}_k)}{\phi(\mathbf{r}_k)} + \frac{\nabla_k \phi(\mathbf{r}_k)}{\phi(\mathbf{r}_k)}\left(\sum_{j\neq k}\frac{(\mathbf{r}_k - \mathbf{r}_j)}{r_{kj}}u'(r_{kj})\right) \tag{69}$$

Differentiating the three factors in the second term of (68),

$$\nabla_k \phi(\mathbf{r}_k)\left[\prod_{i\neq k}\phi(\mathbf{r}_i)\right]\exp\left(\sum_{i<j}u(r_{ij})\right)\sum_{j\neq k}\nabla_k u(r_{kj}) \quad + \tag{70}$$

$$\prod_i \phi(\mathbf{r}_i)\exp\left(\sum_{i<j}u(r_{ij})\right)\left[\sum_{i\neq k}\nabla_k u(r_{ki})\sum_{j\neq k}\nabla_k u(r_{kj}) + \sum_{j\neq k}\nabla_k^2 u(r_{kj})\right] \tag{71}$$

The first term in the square brackets results in

$$\sum_{i,j\neq k}(\frac{(\mathbf{r}_k - \mathbf{r}_i)(\mathbf{r}_k - \mathbf{r}_j)}{r_{ki}r_{kj}}u'(r_{ki})u'(r_{kj}) \tag{72}$$

To compute the second term in square brackets we define

$$M = \nabla_k u(r_{kj}) = \frac{(\mathbf{r}_k - \mathbf{r}_j)}{r_{kj}}u'(r_{kj}) \tag{73}$$

Then

$$\nabla_k^2 u(r_{kj}) = \frac{\partial M}{\partial r_{kj}} = \frac{\partial}{\partial r_{kj}}\left[\frac{(\mathbf{r}_k - \mathbf{r}_j)}{r_{kj}}\right]u'(r_{kj}) + \frac{(\mathbf{r}_k - \mathbf{r}_j)}{r_{kj}}u''(r_{kj}) \tag{74}$$

$$= \frac{(\mathbf{r}_k - \mathbf{r}_j)(\mathbf{r}_k - \mathbf{r}_j)}{r_{kj}^2}\left(u''(r_{kj}) + \frac{2}{r_{kj}}u'(r_{kj})\right) \tag{75}$$

$$= u''(r_{kj}) + \frac{2}{r_{kj}}u'(r_{kj}) \tag{76}$$

$$\tag{77}$$

The analytic expression for the Laplacian of the trial wavefunction is thus

$$\frac{1}{\Psi_T(\mathbf{R})}\nabla_k^2 \Psi_T(\mathbf{R}) = \frac{\nabla_k^2 \phi(\mathbf{r}_k)}{\phi(\mathbf{r}_k)} + \frac{\nabla_k \phi(\mathbf{r}_k)}{\phi(\mathbf{r}_k)}\sum_{j\neq k}\frac{(\mathbf{r}_k - \mathbf{r}_j)}{r_{kj}}u'(r_{kj}) \quad + \tag{78}$$

$$\sum_{i,j\neq k}(\frac{(\mathbf{r}_k - \mathbf{r}_i)(\mathbf{r}_k - \mathbf{r}_j)}{r_{ki}r_{kj}}u'(r_{ki})u'(r_{kj}) + \sum_{j\neq k}\left(u''(r_{kj}) + \frac{2}{r_{kj}}u'(r_{kj})\right) \tag{79}$$

We need the first and second derivative of $u(r_{kj}) = \ln f(r_{kj}) = \ln(1 - a/r_{kj})$.

$$u'(r_{kj}) = \frac{1}{1 - a/r_{kj}} \frac{a}{r_{kj}^2} = \frac{a}{r_{kj}(r_{kj} - a)} \tag{80}$$

and

$$u''(r_{kj}) = \frac{a(a - 2r_{kj})}{r_{kj}^2 (r_{kj} - a)^2} \tag{81}$$

We now have everything we need to obtain the full expression, except the gradient and Laplacian of $\phi(\mathbf{r})$, which is

$$\frac{\nabla_k^2 \phi(\mathbf{r}_k)}{\phi(\mathbf{r}_k)} = 2a(2ax_k^2 - 1) + 2a(2ay_k^2 - 1) + 2a(2a\beta^2 z_k^2 - \beta) \tag{82}$$

$$= 2a(2a(x_k^2 + y_k^2 + \beta z_k^2) - 2 - \beta) \tag{83}$$

and

$$\frac{\nabla_k \phi(\mathbf{r}_k)}{\phi(\mathbf{r}_k)} = -2a(x, y, \beta z) \tag{84}$$

What remains:

- Error analysis with blocking, only for system 2? Before or after optimization?
- Run $N = 10, 50, 100$ for system 2, compare with system 1. Run for several *alpha*.
- Run steepest descent method on system 2 to optimize, then calculate resulting energy
- Plot one-body density with optimal alpha for system 1 and 2.
- Rearrange report: Move analytical results to appendix?
- Write about benchmarks
- Rewrite introduction and abstract
- Write text on results
- Add reference, the same as in the project description
- Look through whole report

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).