Variational Monte Carlo studies of bosonic systems

Anna Stray Rongve Knut Magnus Aasrud
Amund Midtgard Raniseth

March 28, 2021

Abstract

1 Introduction

Finding the ground state of a confined Bose-gas can be a difficult task to do analytically. As such, we shall implement a variational Monte Carlo solver specialized for the problem at hand, and solve it numerically. This is done in Rust, a fast, safe and modern language.

We will first introduce the theory behind our approach and find an analytical solution for a simplified system. This is done to have a benchmark from which we can compare the performance of our numerical solver. Thereafter we present the methodology behind our codebase and how its implemented. The sections thereafter show our findings and a discussion regarding them.

2 Theory

The system in question is a hard sphere Bose gas located in a potential well. The potential is an *elliptical harmonic trap*, described for each particle by

$$V_{\text{ext}}(\mathbf{r}) = \frac{1}{2}m\left(\omega_{\text{ho}}^2(r_x^2 + r_y^2) + \omega_z^2 r_z^2\right). \tag{1}$$

Here, $\mathbf{r}$ is the position of the particle and ω_{ho} is the frequency of the trap. Note that setting $\omega_{\text{ho}} = \omega_z$ results in eq. (1) evaluating to $V_{\text{ext}}(\mathbf{r}) = \frac{1}{2}m\omega_{\text{ho}}^2 r^2$, which represents the *spherical* case of the elliptical harmonic trap. As a simplification, we hereby denote the spherical case as (S) and the general elliptical case as (E).

In addition to this external potential, we represent the inter-boson interactions with the following pairwise, repulsive potential[1]:

$$V_{\text{int}}(|\mathbf{r}_i - \mathbf{r}_j|) = \begin{cases} \infty & |\mathbf{r}_i - \mathbf{r}_j| \leq a \\ 0 & |\mathbf{r}_i - \mathbf{r}_j| > a \end{cases}, \tag{2}$$

where a is the hard-core diameter of the bosons. Eq. (1) and eq. (2) evaluate to the following two-body Hamiltonian:

$$H = \sum_i^N \left(-\frac{\hbar^2}{2m} \nabla_i^2 + V_{\text{ext}}(\mathbf{r}_i) \right) + \sum_{i<j}^N V_{\text{int}}(|\mathbf{r}_i - \mathbf{r}_j|). \tag{3}$$

The term $-\frac{\hbar^2}{2m}\nabla_i^2$ stems from the kinetic energy of the system and the index notation used is described in A.2.1. By scaling into length units of a_{ho} and energy units of $\hbar\omega_{\text{ho}}$, this equation is further simplified into:

$$H = \frac{1}{2} \sum_i^N \left(-\nabla_i^2 + r_{x,i}^2 + r_{y,i}^2 + \gamma^2 r_{z,i}^2 \right) + \sum_{i<j}^N V_{\text{int}}(|\mathbf{r}_i - \mathbf{r}_j|), \tag{4}$$

where $\gamma = \frac{\omega_z}{\omega_{\text{ho}}}$. The derivation of (4) is explained in A.3.5. Lastly we also define the so-called local energy, which is the quantity we want to integrate over to find the total energy of the system:

$$E_L(\mathbf{r}) = \frac{1}{\Psi_T(\mathbf{r})} H \Psi_T(\mathbf{r}) \tag{5}$$

2.1 The variational principle

Given the above Hamiltonian, we can introduce the concept of a *trial wave function* $\Psi_T(\alpha)$. This is a normalized ansatz to the ground state wave function, parametrized by the parameter(s) α. This gives us a way of deploying the *variational principle* by varying said parameter α to our needs:

We know that for any normalized function Ψ_T, the expected energy is higher than the ground state energy (as proved in [2] on p. 293-294), viz.

$$\langle E(\alpha) \rangle = \langle \Psi_T(\alpha) | H | \Psi_T(\alpha) \rangle \geq E_0 = \langle \Psi_0 | H | \Psi_0 \rangle. \tag{6}$$

Thus, minimizing over α will give an approximation of the true ground state (perhaps even an accurate answer).

Evaluating this integral is computationally demanding. Hence, we utilize Monte Carlo integration to allow scalability. This is done by changing the particles positions where the shifting follows some rules. For each change, the local energy is sampled resulting in an expectation value of the energy $\langle E \rangle$ for the Hamiltonian.

To find the lowest value with regards to α, we could either test over many different values, or use gradient descent methods. The latter requires an expression for $\frac{\partial E}{\partial \alpha}$, which we choose to define thusly:

$$\dot{E}_\alpha = \frac{\partial \langle E_L(\alpha) \rangle}{\partial \alpha}.$$

Using the additional notation of $\dot{\Psi}_{T,\alpha} = \frac{\partial \langle \Psi_T(\alpha) \rangle}{\partial \alpha}$, it can be shown that by using the chain rule and the hermiticity of the Hamiltonian [3], we get the expression

$$\dot{E}_\alpha = 2 \left(\left\langle \frac{\dot{\Psi}_{T,\alpha}}{\Psi(\alpha)} E_L(\alpha) \right\rangle - \left\langle \frac{\dot{\Psi}_{T,\alpha}}{\Psi(\alpha)} \right\rangle \langle E_L(\alpha) \rangle \right) \tag{7}$$

Further explanation on how this is used in our gradient descent method is explained in the section Steepest gradient descent.

2.2 Wave function

For N particles, we use the following trial wave function:

$$\Psi_T(\mathbf{r}_1, ..., \mathbf{r}_N, \alpha, \beta) = \prod_i g(\alpha, \beta, \mathbf{r}_i) \prod_{j<k} f(a, |\mathbf{r}_j - \mathbf{r}_k|) \tag{8}$$

Once again, the index notation is described in A.2.1. Here we've used that

$$g(\alpha, \beta, \mathbf{r}_i) = e^{-\alpha(x_i^2 + y_i^2 + \beta z_i^2)},$$
$$\text{and } f(a, |\mathbf{r}_i - \mathbf{r}_j|) = \begin{cases} 0 & |\mathbf{r}_i - \mathbf{r}_j| \leq a \\ 1 - \frac{a}{|\mathbf{r}_i - \mathbf{r}_j|} & |\mathbf{r}_i - \mathbf{r}_j| > a \end{cases},$$

as shown in [1]. Simplifying the trial wave function can prove useful, in order to reduce the number of floating point operations. An analytical expression is also convenient for comparison with the numerical calculations.

2.3 Importance sampling

Importance sampling, compared to the brute force Metropolis sampling, sets a bias on the sampling, leading it on a better path. This means that the desired standard deviation is acquired after fewer Monte Carlo cycles.

For our quantum mechanical scenario with boson particles in a magnetic trap, the bias has its root in the so-called quantum force. This quantum force pushes the walker (the boson particle) to the regions where the trial wave function is large. It is clear that this yields a faster convergence compared to the Metropolis algorithm, where the walker has the same probability of moving in all directions.

The quantum force $\mathbf{F}$ is given by the formula

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

which is derived from the Fokker-Planck equation, using the Langevin equation to generate the next step with Euler's method, and by making the probability density converge to a stationary state.

2.3.1 Fokker-Planck

For one particle (or walker), the one-dimensional Fokker-Planck equation for a diffusion process is:

$$\frac{\partial P}{\partial t} = D\frac{\partial}{\partial x}\left(\frac{\partial}{\partial x} - F\right)P(x,t)$$

Where $P(x,t)$ is a time-dependent probability density, D is the diffusion coefficient and F is a drift term which in our case is driven by the quantum force.

2.3.2 Langevin equation

The Langevin equation solution gives the position of the walker in the next timestep. The Langevin equation is:

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

Converting this to a function yielding the new position y in a computational manner, we use Euler's method.

$$y = x + DF(x)\Delta t + \xi\sqrt{\Delta t}. \tag{9}$$

Here x is the old position, y is the new position and ξ is a randomly sampled value from the normal distribution. In scaled units, the diffusion coefficient evaluates to $\frac{1}{2}$. The timestep Δt has stable values within the range $\Delta t \in [0.001, 0.01]$, so we'll simply choose the value $\Delta t = 0.005$ here.

2.3.3 Fokker-Planck and Langevin equation in importance sampling

In order to use these equations for our importance sampling, we start with the original Fokker-Planck equation.

After inserting D as the diffusion coefficient and $\mathbf{F_i}$ as component i of the drift velocity, we can make the probability density converge to a stationary state by setting its partial derivative over time to zero.

$$\frac{\partial P}{\partial t} = \sum_i D \frac{\partial}{\partial \mathbf{x_i}} \left(\frac{\partial}{\partial \mathbf{x_i}} - \mathbf{F_i} \right) P(\mathbf{x}, t)$$

Where then $\frac{\partial P}{\partial t} = 0$, and by expanding the parenthesis and moving the double partial derivative over to the other side, we obtain:

$$\frac{\partial^2 P}{\partial \mathbf{x_i}^2} = P \frac{\partial}{\partial \mathbf{x_i}} \mathbf{F_i} + \mathbf{F_i} \frac{\partial}{\partial \mathbf{x_i}} P$$

By inserting $g(\mathbf{x}) \frac{\partial P}{\partial x}$ for the drift term, $\mathbf{F}$, we get

$$\frac{\partial^2 P}{\partial \mathbf{x_i}^2} = P \frac{\partial g}{\partial P} \left(\frac{\partial P}{\partial \mathbf{x}_i} \right)^2 + P g \frac{\partial^2 P}{\partial \mathbf{x}_i^2} + g \left(\frac{\partial P}{\partial \mathbf{x}_i} \right)^2$$

Where again the left hand side can be set to zero to comply with the fact that at a stationary state, the probability density is the same for all walkers.

For this to be solvable, the remaining terms have to cancel each other. This is only possible when $g = P^{-1}$, which gives the aformentioned quantum force, $\mathbf{F}$,

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

From here, The Green's function is deployed as

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

Which will be part of the proposal distribution, $q(y, x)$ as

$$q(y, x) = \frac{G(x, y, \Delta t) \left| \Psi_T(y) \right|^2}{G(y, x, \Delta t) \left| \Psi_T(x) \right|^2} \tag{10}$$

2.4 Analytical derivations

2.4.1 Local energy simple Gaussian wave function

As a test case to be compared against our numerical implementation, we want to find an analytical expression for the energy of the trial wave function. We simplify by studying only the non-interacting part, which is done by setting the parameter $a = 0$. We also set $\beta = 1$, giving us the following trial wave function:

$$\Psi_T(\mathbf{r_1}, \mathbf{r_2}, \dots, \mathbf{r_N}, ,) = \prod_i \exp(-\alpha r_i^2).$$

Considering (5):

$$\begin{aligned} E_L(\mathbf{r}) &= \frac{1}{\Psi_T(\mathbf{r})} H\Psi_T(\mathbf{r}) = \frac{1}{\Psi_T(\mathbf{r})}\left[\sum_i^N \left(\frac{-\hbar^2}{2m}\nabla_i^2 + V_{\text{ext}}(\mathbf{r}_i)\right)\right]\Psi_T(\mathbf{r}) \\ &= \frac{1}{\Psi_T(\mathbf{r})}\left[\sum_i^N \left(\frac{-\hbar^2}{2m}\nabla_i^2\Psi_T(\mathbf{r}) + V_{\text{ext}}(\mathbf{r}_i)\Psi_T(\mathbf{r})\right)\right]. \end{aligned}$$

We simplify $\nabla_i^2\Psi_T$ as shown in A.3.1, yielding

$$\nabla^2\Psi_t(\mathbf{r}) = -2\alpha\Psi_T\left(\dim - 2\alpha\mathbf{r}_i^2\right), \tag{11}$$

where dim is the dimension of the system (1, 2 or 3). Given eq. (11), we find that the local energy for N particles in the case of the simple Gaussian wavefunction is

$$E_L(\mathbf{r}) = \frac{\hbar^2}{m}\alpha N \dim + \left(\frac{1}{2}m\omega_{\text{ho}}^2 - 2\alpha^2\right)\sum_i^N \mathbf{r}_i^2, \tag{12}$$

as shown in A.3.2. We can simplify this even further by scaling, namely setting $\hbar = m = 1$, which gives us the equation

$$E_L(\mathbf{r}) = N\alpha \dim + \left(\frac{1}{2}m\omega_{\text{ho}}^2 - 2\alpha^2\right)\sum_i^N \mathbf{r}_i^2 \tag{13}$$

An even simpler analytic expression is obtained by setting $\omega_{\text{ho}} = 1$ and taking the derivate of the local energy with respect to r_i, giving $\alpha = 0.5$.

$$E_L = \frac{N \dim}{2}$$

2.4.2 Drift force

The following expression for the drift force will be used to **explanation**

$$F = \frac{2\nabla_k \Psi_T(\mathbf{r})}{\Psi_T(\mathbf{r})} = -4\alpha \mathbf{r}_k$$

applying the gradient operator to the trial wavefunction is already shown (appendix: Second derivative of trial wave function).

2.4.3 Local energy for full wave function

With $\beta \neq 0$ and a > 0 the wave function becomes a bit more complicated as the potential/Gaussian can be can now be elliptical and the wave function contains the Jastrow factor. The energy is given as:

$$E(\mathbf{r}) = \frac{1}{\Psi_T(\mathbf{r})} \sum_i^N \nabla_i^2 \Psi_T(\mathbf{r}),$$

To simplify coming equations, we set $\phi(\mathbf{r}) = g(\alpha, \beta, \mathbf{r})$, $u(r_{ij}) = \ln f(r_{ij})$ and $r_{ij} = |r_i - r_j|$. With eq. (8), this results in

$$\Psi_T(\mathbf{r}) = \prod_i^N \phi(\mathbf{r}_i) \exp\left(\sum_{i<j} u(r_{ij})\right)$$

Using this simplification, we show in A.3.3 that the gradient for the k-th particle is equal to:

$$\begin{aligned}\nabla_k \Psi_T(\mathbf{r}) = \nabla_k \phi(\mathbf{r}_4 k) \left[\prod_{i\neq k}^N \phi(\mathbf{r}_k)\right] \exp\left(\sum_{j<m}^N u(r_{jm})\right) \\ + \left[\prod_i^N \phi(\mathbf{r}_i)\right] \exp\left(\sum_{j<m}^N u(r_{jm})\right) \sum_{l\neq k}^N \nabla_k (r_{kl}).\end{aligned}$$

Furthermore, using the resulting Laplacian found in A.3.4, we can find

$$
\begin{aligned}
\frac{1}{\Psi_T(\mathbf{r})}\nabla_k^2\Psi_T(\mathbf{r}) = \frac{\nabla_k\phi(\mathbf{r}_k)}{\phi(\mathbf{r}_k)} + 2\frac{\nabla_k\phi(\mathbf{r}_k)}{\phi(\mathbf{r}_k)}\sum_{j\neq k}\frac{\mathbf{r}_j-\mathbf{r}_k}{\mathbf{r}_{jk}}u'(r_{lk}) \\
+\sum_{j\neq k}\sum_{l\neq k}\frac{\mathbf{r}_j-\mathbf{r}_k}{\mathbf{r}_{jk}}u'(r_{lk}) \\
+\sum_{j\neq k}\sum_{l\neq k}\frac{\mathbf{r}_j-\mathbf{r}_k}{\mathbf{r}_{jk}}\frac{\mathbf{r}_l-\mathbf{r}_k}{\mathbf{r}_{lk}}u'(r_{jk})u'(r_{lk}) \\
+\sum_{l\neq k}\frac{2}{r_{lk}}u'(r_{lk})+u''(r_{lk})
\end{aligned}
$$

where these hold:

$$
\frac{\nabla_k\phi(\mathbf{r}_k)}{\phi(\mathbf{r}_k)} = -2\alpha\begin{bmatrix} x_k^2 \\ y_k^2 \\ \beta z_k^2 \end{bmatrix},
$$

$$
\frac{\nabla_k^2\phi(\mathbf{r}_k)}{\phi(\mathbf{r}_k)} = 2\alpha(2\alpha)[x_k^2+y_k^2+\beta^2 z_k^2]-2-\beta),
$$

$$
u'(r_{ij}) = \frac{r_{ij}}{r_{ij}-a}, \quad \text{for} \quad r_{ij}>a,
$$

$$
u''(r_{ij}) = \frac{a(a-2r_{ij})}{r_{ij}^2(a-r_{ij})^2}, \quad \text{for} \quad r_{ij}>a.
$$

3 Method

3.1 Variational Monte Carlo

Metropolis sampling

Since normalizing the wave function is computationally demanding, we need another way of drawing samples from it. This is where the Metropolis algorithm comes in.

For a given distribution $p(\theta)$, we know that:

$$
p(\theta) \propto g(\theta),
$$

where our goal is to sample from $p(\theta)$. The Metropolis algorithm proceeds as follows:

1. Select an initial value θ_0. This is often chosen randomly.
2. To produce a new sample:
    1. Draw a candidate θ' from the proposal distribution $q(\theta'|\theta_{i-1})$
    2. Compute the ratio $r = \frac{g(\theta')q(\theta_{i-1}|\theta')}{g(\theta_{i-1})q(\theta'|\theta_{i-1})}$
    3. Decide:
        - If $r \geq 1$ or $r > u$ (where u is a random sample from the uniform distribution), set $\theta_i = \theta'$.
        - Else, set $\theta_i = \theta_{i-1}$
3. Repeat as many times as needed.

Our distribution p here is of course Ψ_T. The proposal distribution $q(\theta'|\theta_{i-1})$ is the distribution that suggests a new sample given the previous one. In the case which we'll call the *brute force Metropolis*, our choice here is the normal distribution centered around the previous sample to favor samples close to it. This makes the sequence into a random walk and our r becomes a bit simpler, namely $r = \frac{g(\theta')}{g(\theta_{i-1})}$. A flaw with this is that the sampler might not converge around the important parts[1] and jump around a bit "willy-nilly." To combat this, we utilize the proposal distribution shown in eq. (10) to get more relevant samples quicker. We will use the term *importance sampling Metropolis* to refer to this method.

Monte Carlo integration

To evaluate the required integrals and find the energy, we use Monte Carlo integration (see section 2.3.1 of our previous work [4]). Instead of sampling randomly, we use the Metropolis algorithm as explained above to get our new samples.

Steepest gradient descent

Lastly, to reach the optimum value of α, we wish to find the minimum of $E(\alpha)$, in tune with the variational principle as shown in 2.1. This is achieved using a simple steepest gradient descent (or SDG) method. Briefly explained, it works by following the negative value of the gradient, which always points in the direction of greatest momentaneous descent. So it proceeds as follows:

$$\alpha_{i+1} = \alpha_i - \eta \dot{E}_\alpha,$$

where $\dot{E}_\alpha$ is the gradient of the energy with regards to α as defined in (7) and η is the so-called *learning rate* - a value which decides how big of a leap we want to do in the direction of the negative gradient.

[1] Namely the greater values of the distribution, which actually contribute.

Numerical differentiation

To numerically calculate the Laplacian (for use in evaluating the kinetic energy), we use the second order central difference approximation, namely

$$\frac{d^2 f(x)}{dx^2} \approx \frac{f(x-h) - 2f(x) + f(x+h)}{h^2},$$

for sufficiently small h.

3.2 Statistical analysis

Blocking

All of these computer simulations can be considered "computational experiments," and can thus be statistically analyzed in the same way as real-life experiments. There is one catch, however: All our samples are correlated with the previous one, making a "correlation chain" of sorts. Nilsen [5] presents that in the case of correlated samples, the standard deviation of a sampled quantity (in our case E) is

$$\sigma_E = \sqrt{\frac{1 + 2\tau/\Delta t}{n-1}\left(\langle E^2\rangle - \langle E\rangle^2\right)},$$

where Δt is the time between each sample and τ is the time between one sample and the next uncorrelated sample - called the *correlation time*. To combat this correlation effect, we need to split our samples into blocks, each containing N_{blocking} samples. Assuming the blocks are big enough that they are uncorrelated, we can calculate the variance normally based on their mean.

A natural value for N_{blocking} would be τ, but we don't know it's value. A computationally efficient way of finding it is to plot the standard deviation against different values of N_{blocking}. The error will initially increase, but eventually plateau, by which we've reached uncorrelated samples and subsequently our desired value for N_{blocking} [5]. Using this, we can confidently compute the variance of our Monte Carlo integration.

3.3 Natural length scale

As shown in A.3.5, we use scaled length-units of $r \to r' = \frac{r}{a_0}$ and $E \to E' = \frac{E}{\hbar\omega_{\text{ho}}}$. This gives us the constants of $a_0 = \frac{a}{a_{\text{ho}}}$ and $\gamma = \frac{\omega_z}{\omega_{\text{ho}}}$. These scaled units are used throughout the program and are later reversed to get real values.

3.4 Choice of programming language

The Variational Monte Carlo solver is implemented in Rust. The reasons for choosing this language are two-fold:

- Rust is known for being on par with C/C++ in regards to efficiency. This makes it a great fit for heavy numerical computation, which is needed in a task like this.
- In contrast with C/C++, Rust has guaranteed memory safety. This does of course not solve any logical errors in our code, but it alleviates a lot of the memory struggles often met when dealing with such a low-level language - especially with regards to parallellization.

In addition to these, we were all very intrigued by this modern language that is currently taking the computer science world by storm.

3.4.1 Auto-vectorization

Auto-vectorization in Rust is almost as easy as in C++, and can be applied by setting `RUSTFLAGS = "-C opt-level=3 -C target-cpu=native"` in the `Cargo.toml` file, which basically inputs the parameters to the compiler at compiletime. The first flag tells the compiler to run all possible optimizations. Setting `opt-level=2` is the same as running the alias `-O` which only runs some optimizations [6]. `target-cpu` tells the compiler which cpu to compile specific code for. By inserting `native`, the compiler will compile for the cpu the compiler is run at [6].

However, simple loops like `for i in 0..n` will not be properly vectorized due to the fact that the compiler cannot guarantee that the length of the loop is within bounds of the slice iterated over. The easiest way to ensure that this does not happen is to use an iterator. If this cannot be done, hinting to LLVM the length of the slice would also eliminate the bound checks. An example is to define the slice as `let x = &x[0..n];`.

4 Results

4.1 Finding the optimal α

Using the brute force Metropolis algorithm, we calculated the expected value of the local energy at different values of α. This was also done at different dimensions and number of particles. The simulation over all these variables were done once for each core of the processor running them. In our case, this resulted in 8 runs. The mean over all runs are seen in figure 1.

In figure 1, we see that the optimal value of α seems to be consistently on the value 0.5, as expected. However, for $N = 100$ the mean deviates a bit from our expectation. A more telling picture appears when we plot the standard deviation over the CPU cores as a function of α instead of the expected local energy. This is shown in figure 2. From this its much more clear that we're reaching the actual desired value of α at 0.5, regardless of how many number of particles we're simulating for.

5 Discussion

6 Conclusion

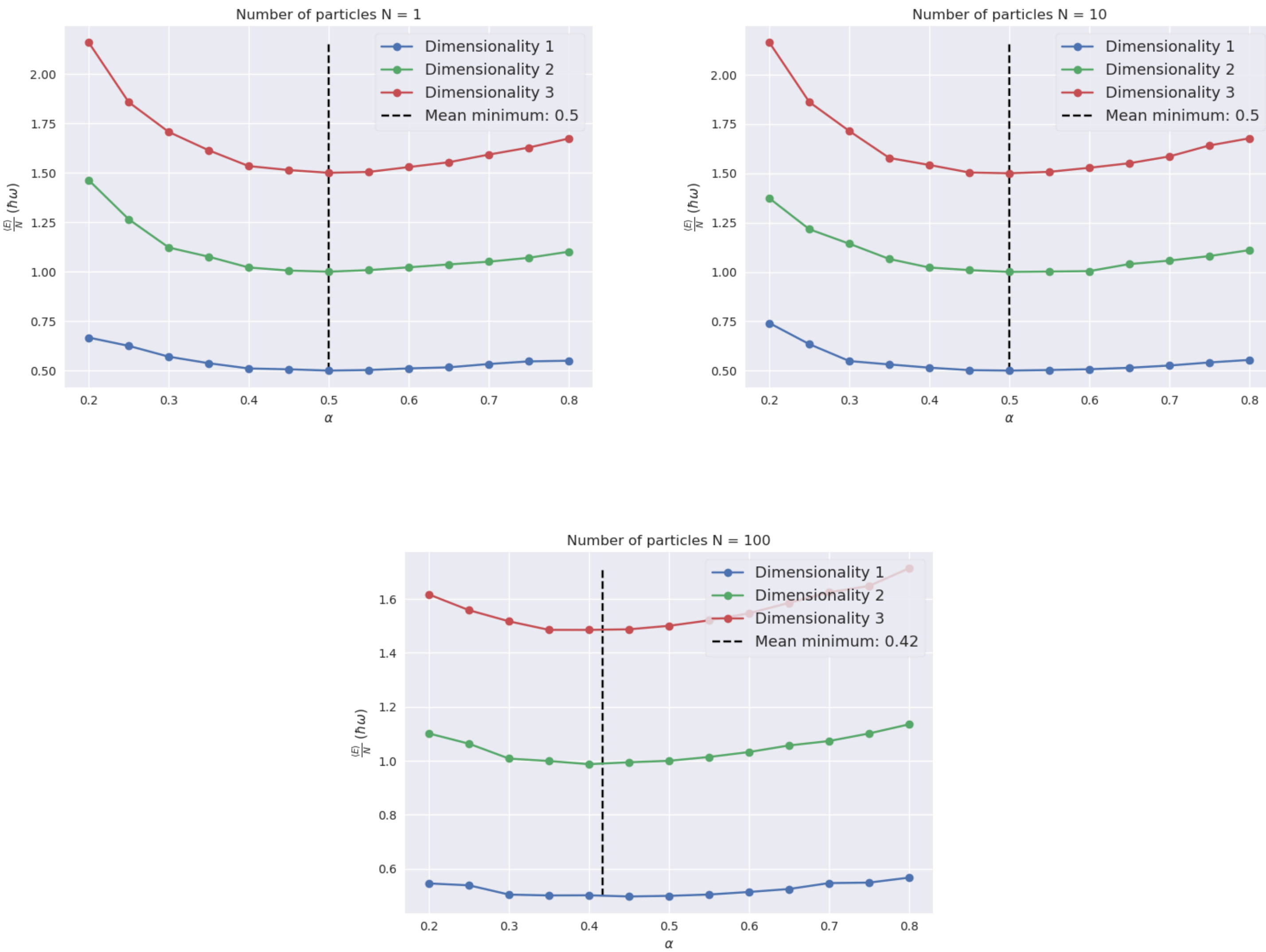

Figure 1: Expected local energy (in units of $\hbar\omega_{\text{ho}}$) per particle, found at $N = 1, 10, 100$ and for $\dim = 1, 2, 3$. The results are the means over simulations run on 8 CPU cores simultaneously. The black dashed line shows the mean minimum over all three dimensions.

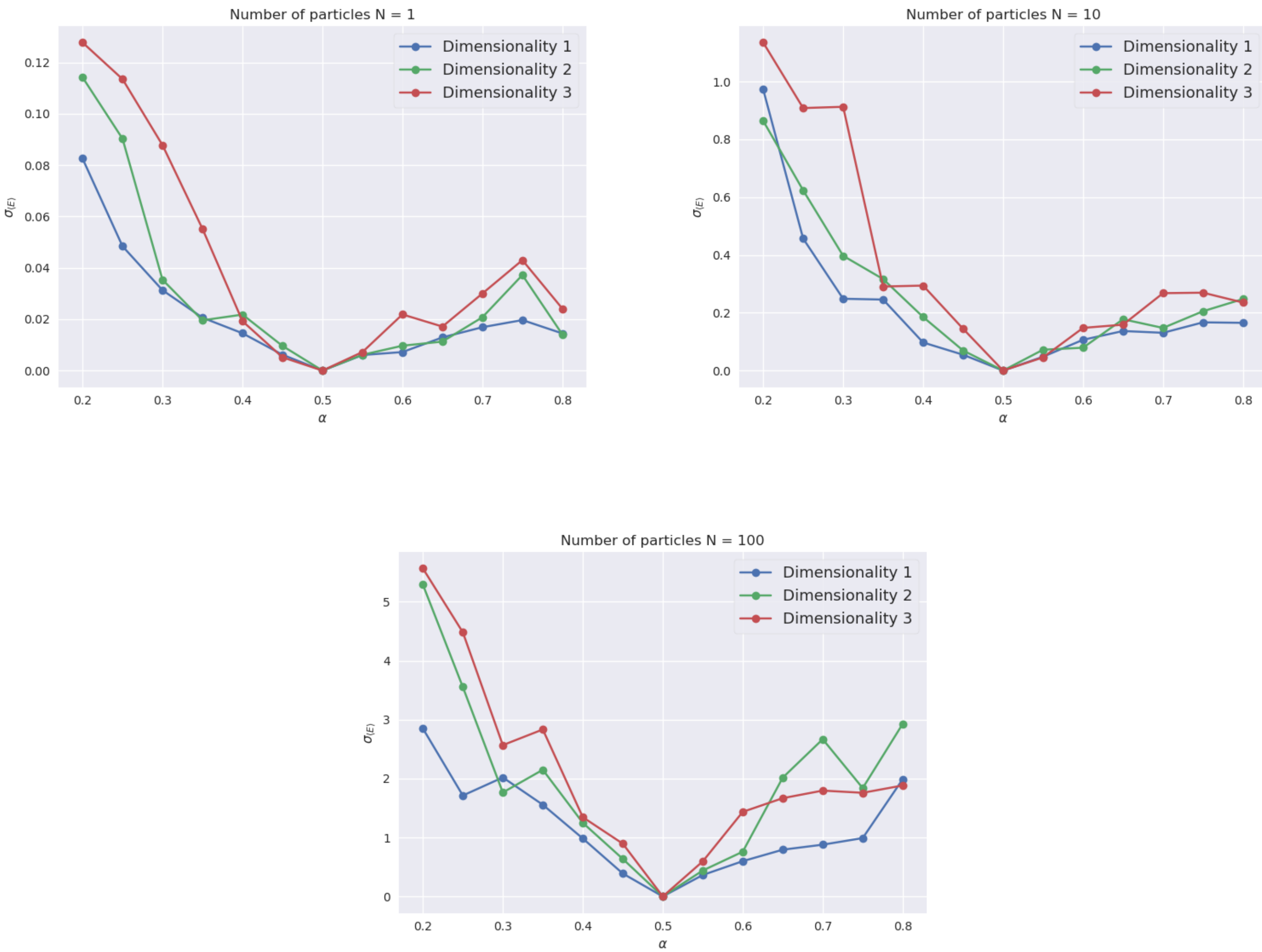

Figure 2: Expected local energy (in units of $\hbar\omega_{\mathrm{ho}}$) per particle, found at $N = 1, 10, 100$ and for $\dim = 1, 2, 3$. The results are the means over simulations run on 8 CPU cores simultaneously. The black dashed line shows the mean minimum over all three dimensions.

A Appendix

A.1 Source code

All source code for both the Rust VMC implementation and this document is found in the following GitHub Repository

https://github.com/kmaasrud/vmc-fys4411

A.2 Notation and other explanations

A.2.1 Index notation for sums and products

For products and sums, the following convention is used:

$$\sum_{i<j}^{N} = \sum_{i=1}^{N} \sum_{j=i+1}^{N}, \quad \text{or} \quad \prod_{i<j}^{N} = \prod_{i=1}^{N} \prod_{j=i+1}^{N}$$

A.3 Calculations

A.3.1 Second derivative of trial wave function

$$\begin{aligned}
\nabla_i^2 \Psi_T(\mathbf{r}) &= \nabla_i \cdot \left[\frac{\partial}{\partial x_i}, \frac{\partial}{\partial y_i}, \frac{\partial}{\partial z_i}\right] \Psi_T(\mathbf{r}) \\
&= \nabla_i \cdot \left[\frac{\partial}{\partial x_i} \exp\left(-\alpha \mathbf{r}_i^2\right), \frac{\partial}{\partial y_i} \exp\left(-\alpha \mathbf{r}_i^2\right), \frac{\partial}{\partial z_i} \exp\left(-\alpha \mathbf{r}_i^2\right)\right] \\
&= \nabla_i \cdot \left[-2\alpha x_i \exp\left(-\alpha \mathbf{r}_i^2\right), -2\alpha y_i \exp\left(-\alpha \mathbf{r}_i^2\right), -2\alpha z_i \exp\left(-\alpha \mathbf{r}_i^2\right)\right] \\
&= -2\alpha \left[\exp\left(-\alpha \mathbf{r}_i^2\right)(1 - 2\alpha x_i^2), \exp\left(-\alpha \mathbf{r}_i^2\right)(1 - 2\alpha y_i^2), \exp\left(-\alpha \mathbf{r}_i^2\right)(1 - 2\alpha z_i^2)\right] \\
&= -2\alpha \Psi_T \left[(1 - 2\alpha x_i^2), (1 - 2\alpha y_i^2), (1 - 2\alpha z_i^2)\right] \\
&= -2\alpha \Psi_T \sum_{d=x,y,z} 1 - 2\alpha d_i^2 \\
&= -2\alpha \Psi_T (\dim - 2\alpha \mathbf{r}_i^2)
\end{aligned}$$

A.3.2 Local energy for Gaussian wave function

Starting with

$$E_L(\mathbf{r}) = \frac{1}{\Psi_T(\mathbf{r})} \left[\sum_{i}^{N} \left(\frac{-\hbar^2}{2m} \nabla_i^2 \Psi_T(\mathbf{r}) + V_{\text{ext}}(\mathbf{r}_i) \Psi_T(\mathbf{r})\right)\right],$$

and using the result from A.3.1, this results in:

$$
\begin{aligned}
E_L(\mathbf{r}) &= \frac{1}{\Psi_T(\mathbf{r})}\left[\sum_i^N \left(\frac{\hbar^2\alpha}{m}(\dim - 2\alpha\mathbf{r}_i^2) + \frac{1}{2}m\omega_{\text{ho}}^2\mathbf{r}_i^2\right)\Psi_T(\mathbf{r})\right] \\
&= \frac{\hbar^2}{m}\alpha N \dim + \left(\frac{1}{2}m\omega_{\text{ho}}^2 - 2\alpha^2\right)\sum_i^N \mathbf{r}_i^2
\end{aligned}
$$

A.3.3 Gradient of interacting trial wave function

Rewriting the wave function to

$$
\Psi_T(\mathbf{r}) = \left[\prod_i^N \phi(\mathbf{r}_i)\right]\exp\left(\sum_{i<j} u(r_{ij})\right)
$$

where $r_{ij} = |r_i - r_j|$ and we set $u(r_{ij}) = \ln f(r_{ij})$. Lastly $g(\alpha, \beta, \mathbf{r}_i)$ is redefined to the following function

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

For convenience

$$
\Psi_1(\mathbf{r}_i) = \prod_i^N \phi(\mathbf{r}_i)
$$

and

$$
\Psi_2(\mathbf{r}_{ij}) = \exp\left(\sum_{i<j} u(r_{ij})\right)
$$

where Ψ_1 and Ψ_2 is the one-body and correlated part of the wave function, respectively. Both parts have simple dependency of the k'th particle. Ψ_1 is a product of one-body wave functions with only one factor dependent of $\mathbf{r}_k$ and Ψ_2 is $\mathbf{r}_k$ - dependent for the pairs $\sum_{i\neq k} u(\mathbf{r}_{ik})$. Hence the first derivatives becomes

$$
\nabla_k \Psi_1(\mathbf{r}) = \left[\prod_{i\neq k}^N \phi(\mathbf{r}_i)\right]\nabla_k \phi(\mathbf{r}_k)
$$

$$
\nabla_k \Psi_2(\mathbf{r}_{ij}) = \exp\left(\sum_{i<j} u(r_{ij})\right)\sum_{i\neq k}\nabla_k u(\mathbf{r}_{ik})
$$

Giving the first derivate of the trail wave function

$$\nabla_k \Psi_T(\mathbf{r}) = \nabla_k \phi(\mathbf{r}_k) \left[\prod_{i \neq k}^{N} \phi(\mathbf{r}_i) \right] \exp \left(\sum_{i<j} u(r_{ij}) \right) + \prod_{i}^{N} \phi(\mathbf{r}_i) \exp \left(\sum_{i<j} u(r_{ij}) \right) \sum_{i \neq k} \nabla_k u(\mathbf{r}_{ik})$$

A.3.4 Laplacian of interacting trial wave function

The Laplacian of the wacefunction needs to be evaluated in order to calculate

$$\frac{1}{\Psi_T(\mathbf{r})} \nabla_k \nabla_k \Psi_T(\mathbf{r})$$

The last part, $\nabla_k \Psi_T(\mathbf{r})$ is calculated in the section above / equation (**Reference here**). Next step is then to calculate

$$\begin{aligned}
\nabla_k \nabla_k \Psi_T(\mathbf{r}) =& \nabla_k \Bigg(\nabla_k \phi(\mathbf{r}_k) \left[\prod_{i \neq k} \phi(\mathbf{r}_i) \right] \exp \left(\sum_{j<m} u(r_{jm}) \right) \\
&+ \prod_{i} \phi(\mathbf{r}_i) \exp \left(\sum_{j<m} u(r_{jm}) \right) \sum_{l \neq k} \nabla_k u(\mathbf{r}_{kl}) \Bigg) \\
\nabla_k \nabla_k \Psi_T(\mathbf{r}) =& \prod_{i \neq k} \left[\nabla_k^2 \phi(\mathbf{r}_k) \exp \left(\sum_{j<m} u(r_{jm}) \right) + \nabla_k \phi(\mathbf{r}_k) \cdot \nabla_k \exp \left(\sum_{j<m} u(r_{jm}) \right) \right] \\
&+ \nabla_k \prod_{i} \phi(\mathbf{r}_i) \exp \left(\sum_{j<m} u(r_{jm}) \right) \sum_{l \neq k} \nabla_k u(r_{kl}) \\
&+ \nabla_k \exp \left(\sum_{j<m} u(r_{jm}) \right) \prod_{i} \phi(\mathbf{r}_i) \sum_{l \neq k} \nabla_k u(r_{kl}) \\
&+ \nabla_k \sum_{l \neq k} \nabla_k u(r_{kl}) \prod_{i} \phi(\mathbf{r}_i) \exp \sum_{j<m} u(r_{jm})
\end{aligned}$$

In order to avoid writing long calculations, the three main gradients are calculated below. The last of the three following expressions/equations is a bit more of a hazard to calculate. First the product rule is used. Then a rule for the gradient is applied where the gradient of a unit vector is 2 divided by its magnitude. u' is parallel to the unit vector, hence their product becomes a scalar, the second derivate of u.

$$\nabla_k \exp \left(\sum_{j<m} u(r_{jm} \right) = \exp \left(\sum_{j<m} u(r_{jm} \right) \sum_{l \neq k} \nabla_k u(r_{kl})$$

$$\nabla_k \prod_i \phi(\mathbf{r}_i) = \prod_{i \neq k} \phi(\mathbf{r}_i) \nabla_k \phi(\mathbf{r}_k)$$

$$\begin{aligned}
\nabla_k \sum_{l \neq k} \nabla_k u(r_{kl}) &= \sum_{l \neq k} \nabla_k \left(\frac{\mathbf{r}_l - \mathbf{r}_k}{\mathbf{r}_{lk}} u'(r_{lk}) \right) \\
&= \sum_{l \neq k} \left(\nabla_k \frac{\mathbf{r}_l - \mathbf{r}_k}{\mathbf{r}_{lk}} u'(r_{lk}) + \frac{\mathbf{r}_l - \mathbf{r}_k}{\mathbf{r}_{lk}} \nabla_k u'(r_{lk}) \right) \\
&= \sum_{l \neq k} \frac{2}{r_{lk}} + u''(r_{lk})
\end{aligned}$$

Finally the Laplacian can be calculated, by reintroducing the fraction $\frac{1}{\Psi_T(\mathbf{r})}$

$$\begin{aligned}
\frac{1}{\Psi_T(\mathbf{r})} \nabla_k^2 \Psi_T(\mathbf{r}) =& \frac{\prod_{i \neq k} \phi(\mathbf{r}_i)}{\prod_i \phi(\mathbf{r}_i)} \left(\nabla_k^2 \phi(\mathbf{r}_k) + \nabla_k \phi(\mathbf{r}_k \sum_{l \neq k} \nabla_k u(r_{kl}) \right) + \left(\frac{\nabla_k \phi(\mathbf{r}_i)}{\phi(\mathbf{r}_i)} \sum_{l \neq k} \nabla_k u(r_{kl}) \right) \\
&+ \sum_{l \neq k} \nabla_k u(r_{kl}) + \sum_{j \neq k} \nabla_k u(r_{kj}) + \nabla_k \sum_{l \neq k} \nabla_k u(r_{kl})
\end{aligned}$$

The second and third terms are the same. Two of the terms are shown in the calculations above and $\nabla_k u(r_{kl})$ is the unit vector multiplied with the derivate of a scalar. Then we have the final expression

$$\begin{aligned}
\frac{1}{\Psi_T(\mathbf{r})} \nabla_k^2 \Psi_T(\mathbf{r}) =& \frac{\nabla_k \phi(\mathbf{r}_k)}{\phi(\mathbf{r}_k)} + 2 \frac{\nabla_k \phi(\mathbf{r}_k)}{\phi(\mathbf{r}_k)} \sum_{j \neq k} \frac{\mathbf{r}_j - \mathbf{r}_k}{\mathbf{r}_{jk}} u'(r_{lk}) \\
&+ \sum_{j \neq k} \sum_{l \neq k} \frac{\mathbf{r}_j - \mathbf{r}_k}{\mathbf{r}_{jk}} u'(r_{lk}) + \sum_{j \neq k} \sum_{l \neq k} \frac{\mathbf{r}_j - \mathbf{r}_k}{\mathbf{r}_{jk}} \frac{\mathbf{r}_l - \mathbf{r}_k}{\mathbf{r}_{lk}} u'(r_{jk}) u'(r_{lk}) \\
&+ \sum_{l \neq k} \frac{2}{r_{lk}} u'(r_{lk}) + u''(r_{lk})
\end{aligned}$$

A.3.5 Scaling of repulsion Hamiltonian

We have the initial expression for the Hamiltonian, (3). Inserting (1), we get:

$$H = \frac{1}{2} \sum_i^N \left(-\frac{\hbar^2}{m} \nabla_i^2 + m \left(\omega_{\text{ho}}^2 (r_{x,i}^2 + r_{y,i}^2) + \omega_z^2 r_{z,i}^2 \right) \right) + \sum_{i<j}^N V_{\text{int}}(|\mathbf{r}_i - \mathbf{r}_j|).$$

We now introduce the scaled length unit $r' = \frac{r}{a_{\text{ho}}}$ which in turn leads to $\nabla_i'^2 = a_{\text{ho}}^2 \nabla_i^2$.

$$H = \frac{1}{2}\sum_i^N \left(-\frac{\hbar^2}{m a_{\text{ho}}^2}\nabla_i'^2 + m a_{\text{ho}}^2 \left(\omega_{\text{ho}}^2 (r_{x,i}'^2 + r_{y,i}'^2) + \omega_z^2 r_{z,i}'^2 \right)\right) + \sum_{i<j}^N V_{\text{int}}(|\mathbf{r}_i - \mathbf{r}_j|)$$

Inserting the definition of $a_{\text{ho}} = \frac{\hbar}{m\omega_{\text{ho}}}$, we get

$$H = \frac{1}{2}\sum_i^N \left(-\hbar\omega_{\text{ho}}\nabla_i'^2 + \hbar\omega_{\text{ho}} \left((r_{x,i}'^2 + r_{y,i}'^2) + \frac{\omega_z^2}{\omega_{\text{ho}}^2} r_{z,i}'^2 \right)\right) + \sum_{i<j}^N V_{\text{int}}(|\mathbf{r}_i - \mathbf{r}_j|),$$

$$H = \frac{\hbar\omega_{\text{ho}}}{2}\sum_i^N \left(-\nabla_i'^2 + (r_{x,i}'^2 + r_{y,i}'^2) + \gamma^2 r_{z,i}'^2) \right) + \sum_{i<j}^N V_{\text{int}}(|\mathbf{r}_i - \mathbf{r}_j|),$$

where $\gamma = \frac{\omega_z}{\omega_{\text{ho}}}$. We lastly reorganize the above to obtain a scaled Hamiltonian $H' = \frac{H}{\hbar\omega_{\text{ho}}}$ and also make sure to scale the function $V_{\text{int}} \to V_{\text{int}}'$ by transitioning from $a \to a' = \frac{a}{a_{\text{ho}}}$.

$$H' = \frac{1}{2}\sum_i^N \left(-\nabla_i'^2 + r_{x,i}'^2 + r_{y,i}'^2 + \gamma^2 r_{z,i}'^2 \right) + \sum_{i<j}^N V_{\text{int}}'(|\mathbf{r}_i' - \mathbf{r}_j'|). \tag{14}$$

By ensuring that we used scaled length units of $r' = \frac{r}{a_{\text{ho}}}$ and scaled energy units of $E' = \frac{E}{\hbar\omega_{\text{ho}}}$, equation (14) holds. For simplification, we will not use the primed notation outside this derivation.

References

[1] M. Hjorth-Jensen, "Project 1." Jan. 2021.

[2] David. J. Griffiths, *Introduction to Quantum Mechanics*. 2005.

[3] M. Hjorth-Jensen, *Advanced Topics in Computational Physics: Computational Quantum Mechanics*. 2020.

[4] K. M. Aasrud, A. S. Rongve, and A. M. Raniseth, "Project 3." Oct. 2019, [Online]. Available: https://github.com/kmaasrud/gq-mcm-fys3150/blob/master/doc/Project-3_Aasrud-Raniseth-Rongve.pdf.

[5] J. K. Nilsen, “Data blocking - FYS4410 lecture.” 2008, [Online]. Available: https://www.uio.no/studier/emner/matnat/fys/nedlagte-emner/FYS4410/v08/undervisningsmateriale/Material%20for%20Part%20I%20by%20Morten%20HJ/Slides%20from%20Lectures/blocking.pdf.

[6] Rust Community, “The rustc book.” [Online]. Available: https://doc.rust-lang.org/rustc/.