Variational Monte Carlo simulation on Bose-Einstein condensation in magnetic trap

Pavlo Panasiuk, Jonas Gabriel Jonsson

(Dated: April 24, 2023)

We implemented a variant of Markov Chain Monte Carlo (MCMC) using brute force Metropolis, as well as MCMC with importance sampling based on Metropolis-Hastings algorithm, alongside with other software relevant for Variational Monte-Carlo method. The software is ready-to-use for Bose-Einstein Condensate phenomenological simulation in cyllindric magnetic trap, yet it may simulate any nonrelativistic quantum system. Our simulation proves robust for the system of non-interactive bosons, while the smooth transition of system parameters with growth of hard-core radii is observed. We state that the variational parameter, corresponding to exponential multiplier in harmonic oscillator ground state, may be well approximated by the non-interactive case $\alpha = \frac{m\omega}{2\hbar}$. The importance of correlated part within trial wave function is discussed. The results, whenever applicable, are partially benchmarked with simulations performed in [1].

I. INTRODUCTION

In this study simulation and numerical calculation is carried out for a quantum mechanical system of a Bose-Einstein condensate (BEC) in a magnetic trap. BEC is a state of matter typically formed when a dilute gas of bosons reaches a temperature very close to absolute zero. For theory and summary of various experiments see [2].

The method used is Variational Monte Carlo (VMC). This is a Monte Carlo method that applies the variational method in quantum mechanics to find approximations to energy eigenstates. The method builds on using "trial wavefunctions" having one or more parameters, and finding the parameters for which the lowest possible energy expectation values is found [3] ch. 14.

We are using this Monte Carlo method via two algorithms. The first one being the standard brute-force Metropolis algorithm which takes probability of different states into account. The second is the Metropolis Hastings algorithm which uses importance sampling and is a more efficient approach. These are described in more detail under the "Theory and methodology" section II.

We begin the analysis by considering a system of non-interacting particles, for which we can compare the results against an analytical solution. Then we proceed to the more realistic case of interacting particles. All of this builds on the Hamiltonian in which both the potential energy of the trap and particle interactions are encoded. The trial wave function is chosen as a reasonable guess for the specific Hamiltonian. All of these are taken from [1].

Furthermore to find the parameters to the trial wave function we use a optimization method called gradient descent. Error analysis on the estimated energies is carried out using the Blocking Method, based on the work in [4]. This estimates the stochastic error, based on calculated variance during the simulation.

The implementation uses parallelization with OpenMP to speed up calculations, by using multiple CPU cores.

Reproducible code can be found in our GitHub repository [5]. Credits for source code also goes to [6] from which we forked the initial code base, and [4] from which we reused scripts for blocking method data analysis.

II. THEORY AND METHODOLOGY

For our numerical calculations we base all simulations on Markov chain Monte Carlo (MCMC) with the Metropolis algorithm and Metropolis-Hastings algorithm. This is a well known standard approach, whose algorithm is described in [3] section 13.5. We will briefly outline the motivation and procedure here, and focus on the points where we have made specific implementation decisions.

A. Motivation for Monte Carlo methods

Ground states in quantum mechanical systems are of utmost interest for many-body researches. If one neglects the possibility for system to produce excitations, actual energy of the system is whatever minimal energy one can extract with whole family of L^2 wave functions. One, however, requires a robust framework for energy calculation.

One of the central parts of this calculation is to be able to calculate the energy for a given system and wave function, without spending unreasonable amount of time.

For this we use the operator "local energy", commonly used in Variational Monte Carlo, and defined as:

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

where Ψ_T is the trial wave function [3] p. 462.

Computing the expectation value of this local energy requires evaluating an integral of a large number of dimensions - three times the number of particles involved. Monte Carlo methods are known to handle such heavy computations particularly well.

Basically the integral is approximated as a sum over N evaluations, where N is the number of Monte Carlo samples.

As such the energy and energy squared (to be used to calculate variance) are from [3] p.465 computed by:

$$\left\langle \hat{E}_L \right\rangle = \int P(\mathbf{R}) E_L(\mathbf{R}) d\mathbf{R} \approx \frac{1}{N} \sum_i^N E_L(\mathbf{R_i}), \tag{2}$$

and , more generally

$$\langle f \rangle = \int P(\mathbf{R}) f(\mathbf{R}) d\mathbf{R} \approx \frac{1}{N} \sum_i^N f(\mathbf{R_i}), \tag{3}$$

where points sampling is defined in specific way (accounting for probability distribution).

This proves to work well for large N and a representative sample distribution. The distribution is created as a Markov chain, using various randomized rules, that we will cover in more detail. Thus making it all a Markov chain Monte Carlo (MCMC) algorithm [3] ch. 12.

B. More on local energy

The local energy and it's variance are computed repeatedly via MCMC as described above. To make a fair comparison, we have done the calculation both with analytically and numerically calculated Hamiltonians.

Analytic derivations were performed both for non-interacting bosons within spherical harmonic oscillator, as well as for hard-core bosons within cyllindric harmonic oscillator. Here we unite both results in one framework.

Note that even though we only used numeric differentiation for verifying our analytics, it is more generic and is ready-to-use for any system.

While derivations can be found at V, here we present benchmark results only.

First of all, if we neglect interaction potential (that is, $a_{hard-core} \to 0$, local energy will be given by

$$E_L = \left(\frac{1}{2} m\omega^2 - \frac{2\alpha^2\hbar^2}{m} \right) \sum_i^N \sqrt{x_i^2 + y_i^2 + \beta^2 z_i^2}^2 + \\ + \frac{(2+\beta)N\alpha\hbar^2}{m} \tag{4}$$

By visual inspection it is apparent that specific choice $\alpha = \frac{m\omega}{2\hbar}$ turns local energy into flat function, i.e. we correctly deduced trial wave function (since we actually took it from harmonic oscillator). Therefore, any proper VMC for this system should be able to find

$$\alpha = \frac{m\omega}{2\hbar}$$
$$\langle E \rangle = \frac{(2+\beta)N\hbar\omega}{2}, \tag{5}$$

where $(2+\beta)$ equals $\{1; 2; 2+\beta\}$ for $\{1D; 2D; 3D\}$ respectively.

Even though we cannot perform such a simple calculation for the repulsive system, for low packing fraction one should expect results similar to the ones above anyway.

C. Importance sampling

In order to get a more efficient computation, we want the algorithm in MCMC to prefer walking towards more probable states, as dictated by Eq. 3. Therefore we implement the Metropolis-Hastings algorithm, that is, a correction induced by ideas from transport theory.

The actual formulas used to achieve importance sampling are in this case the Fokker-Planck equation and the Langevin equation. What it does is adding a bias to the walker to go to a direction of higher probability. At the same time any direction is still possible, which is important in order to achieve ergodicity, that is, there is possibility to cover all possible paths. On the other hand, this way we have a more computationally efficient algorithm that finds the region with more probable states faster. Both methods eventually converged to a good result, but with importance sampling this tends to happen with fewer cycles.

The algorithm can also be made to satisfy detailed balance [3] p. 405, which is another important requirement, meaning roughly that probabilities for movement are balanced at equilibrium.

For more background on Metropolis-Hastings, ergodicity and detailed balance see [3] pages 382, 400 and 404. More about Fokker-Planck equation and the Langevin equation can be found at [7]. We also heavily base our implementations on the algorithm from this page.

D. Optimization

VMC heavily relies on parameter optimization – we do not only want to find the minimal energy, but also we wish to achieve this minimum with least MC cycles amount.

For the most part, in this work we only tune α parameter (exponential multiplier), yet the codebase is capable to work in many dimensional parameter space.

For the optimization part, we use the well-known gradient descent algorithm, which is an iterative optimization method [8]. It uses the gradient of the function to find a local minimum. It is expected to work well when the function undergoing optimization is convex.

$$\vec{p}_{i+1} = \vec{p}_i - \gamma \vec{\nabla} F(\vec{p}_i), \tag{6}$$

where $F(x)$ is the function undergoing optimization, and γ is what is called learning rate. Usually one fine-tunes the learning rate for each specific task, yet we self-deduce it as

$$\gamma = \max \left[0.1 \left| \frac{\vec{p}_1}{\vec{\nabla} F(\vec{p}_1)} \right|, 1 \right]. \tag{7}$$

This choice is not guaranteed to be perfect, yet it works if one starts sufficiently far ($\sim 0.1 : 0.5$ away) from the minima.

E. Parallelization

The numeric calculations in this setup are computationally heavy, as is often the case with Monte Carlo methods. For good results comparable to [1] one would need to run simulations on many thousands of particles and many millions of iterations. We are settling for a smaller computation, but still some parallellization is in order, to get runtime down to merely a few hours.

The approach we use is using OpenMP to parallelize the work on multiple threads running on a multi core CPU with shared memory on a single PC. The parallel parts of the code covers to loop for which we do the Monte Carlo cycles. For the example of 8 threads, we set up 8 random systems, and let the Monte Carlo machinery work on them independently. After completion we take the average of the resulting observables. Whereas we use optimization, parallel sessions are restarted for each iteration of the gradient descent method.

For all output, the results of all system samplers (one per program thread) are combined. Single values like observables, number of cycles and standard deviations are taken as averages of all samplers.

F. Statistical error estimation and Blocking method

The purpose of the error estimation is to be able to present some confidence interval for the presented results of the energy. Without some kind of error bars the result would not tell much at all. Generally one needs to think of both systematical errors and statistical errors.

We will not concern ourselves with systematical errors in this report, since this is mainly a stochastic computation investigation, on a model we assume to be correct. Systematical errors could exist in the Hamiltonian being wrong, but that could only be addressed by comparing with actual experiments.

The goal is thus to find a good estimate of the statistical error. We use the Blocking method, which is a resampling method made popular by Flyvbjerg and Petersen [9]. This method is beneficial when the number of samples is very large. The computational performance scales much better than for example the Bootstrap method.

The basis for the error analysis is the standard deviation, which is the square root of the variance (second central moment). The standard deviation is calculated as

$$\sigma = \sqrt{\langle E^2\rangle - \langle E\rangle^2} \tag{8}$$

with E and E^2 calculated as in Eq. 2 and Eq. 3 through our MCMC machinery. This means that the standard deviation collected is itself computed using stochastic methods and carry statistical errors of it's own.

Here the resampling comes in, where are large number of energy values are stored and further analysed after the run. The issue is that the observable values are not independent, but carries a so-called auto-correlation. The idea with the Blocking method is roughly to divide the sample set first in blocks of half, then halfs again, and so on. For each division statistical methods are used to estimate the auto-correlation and obtain a value closer to the true standard deviation. For details see [10], [9].

Traditionally this is done by making plots and analyse visually to see what standard deviation value it converges to. But after some work from Marius Jonsson there now exists an automatic method as described in [4]. With this article is provided a script which we are using to calculate the final number for estimated standard deviation.

III. RESULTS AND DISCUSSION

As mentioned in the theory section, we carry out a thorough VMC investigation of many-bosons system contained within harmonic oscillator.

A. Naive calculation

As a first checkpoint, we "rely optimization part" to ourselves. That is, we prepared $\langle E(\alpha)\rangle$ plots for various particle number. In the following $\hbar = m = 1$ units are assumed.

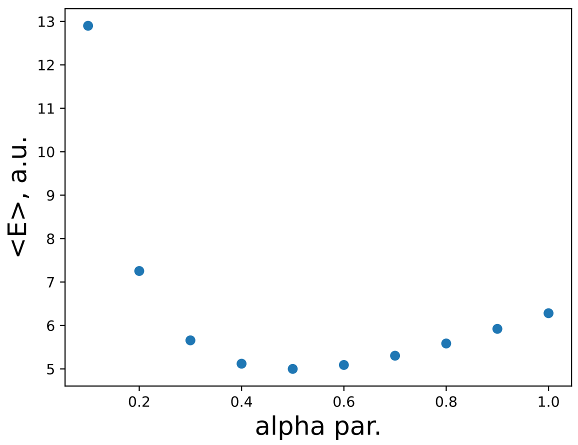

FIG. 1. 1D harmonic oscillator $\omega = 1.0$ for 10 particles. Trial wavefunction is built as a product of single particle ground solutions.

Fig. 1 and Fig. 2 show expected convex behaviour with minima at $\alpha = \frac{m\omega}{2\hbar} = 0.5$. This serves as a partial proof of algorithm's robustness.

Similar calculation for $2D$ and $3D$ gives similar result, with average energy scaling as 2 and $2+\beta$ respectively (see e.g. Fig. 3).

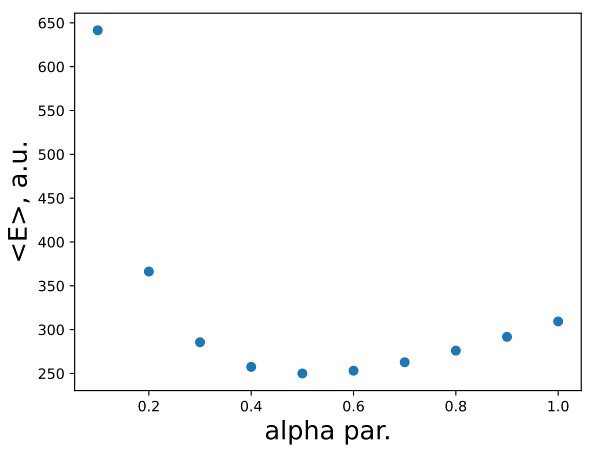

FIG. 2. 1D harmonic oscillator $\omega = 1.0$ for 500 particles. Trial wavefunction is built as a product of single particle ground solutions.

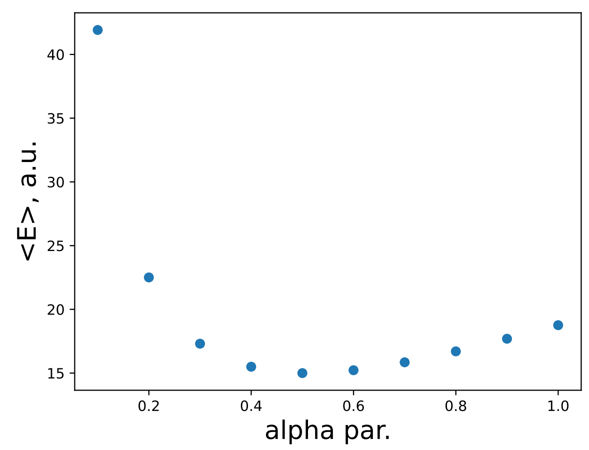

FIG. 3. 3D harmonic oscillator $\omega = 1.0, \beta = 1.0$ for 10 particles. Trial wavefunction is built as a product of single particle ground solutions.

We also compared analytic/numeric implementation for derivatives, both for double check and for CPU difference estimation. For 10 particles in 3D (with 1M cycles), analytic/default implementation shows respectively $\sim 60s/\sim 200s$ on an author's laptop. Result evaluation though proves to coincide in these cases. Further, if not mentioned explicitly, we will use analytic implementation.

B. Importance sampling

In order to compare the Metropolis brute-force algorithm with the Metropolis-Hastings importance sampling algorithm, we've run some benchmark tests.

First we run simulations on the exact solution Gaussian wavefunction, in 3 dimensions, 20 particles, 10^6 cycles and $\alpha = 0.5$.

The Importance sampling algorithm run slightly slower - 2m52s compared to 2m43s for brute force. This can be explained by more complex formulas needing more floating point operations.

Next we compare acceptance rate, which was 98.6% for Importance sampling and 91% for brute force. This is expected, and as argued in [3] ch. 12, higher acceptance rate makes the algorithm more efficient (provided it still complies with ergodicity and detailed balance). We do not expect acceptance rate to be this high for more realistic systems, but this is a good start indicating importance sampling is preferred. Indeed the more complex systems show acceptance rates around 95% and 89% respectively.

A more extensive comparison between these algorithms will be done in section III E, where we look at simulations using optimization and particle interactions. There are some interesting things to note though, that we have observed regarding stability and sensitiveness to values in system setup.

Whereas the brute force simulation readily converged towards good results, the importance sampling simulation initially caused a lot of problems with diverging results. The problem which first was suspected to be a bug in the Monte Carlo algorithm turned out to be caused by the initial state setup. When particles get really close, the quantum force gets huge, thus making the suggested step also large and leading to another improbable region, and therefore most steps rejected. This would in essence make the simulation stuck, and lead to completely unrepresentative energy values, and make optimization process unable to converge.

While in theory it should suffice to require particles in initial state to not overlap, in practice we had to require them to start out slightly further apart. We got the best result by requiring the distance to be larger than four times the so called "hard-core diameter" of the model.

One could argue that our way to set up the initial state is arbitrary but with a large number of cycles, when the simulation reaches a steady state, this choice should not matter much. After careful tweaking of the initial state, then simulations started converging nicely to values very close to the brute force results. The Importance sampling as we will see in III E also shows tendency for lower standard deviation.

Another thing we did during this investigation was to lower the learning rate of the optimization, to make it less sensitive for occasional simulations giving unrepresentative results.

In the final simulations we use a total of 10^6 cycles and step size of 0.1. Experiments have also been done with more cycles and shorter steps. This did not affect the observations above much, and were eventually chosen to fit the performance on available PC machines.

C. Optimization

While the basis is mostly described within Sec. II D, some technicalities remained omitted since then.

The optimization process is limited by 100 iterations, each having number of MC cycles reduced by 100 less than the one provided. While this is highly time-efficient optimization, it is also prone to estimation errors. As a rule, we fine-tune the number of cycles to higher number until the energy reaches numbers comparable with non-interacting case. See Sec. III B for more discussion regarding reliability. The last evaluation, if converged, is performed with the provided number of cycles.

For optimizer's benchmark, see Sec. VII.

D. Repulsive interaction

Introducing two-body repulsive hard-core potential was always a desirable feature for many-body systems. We originate from non-interactive bosonic system to the repulsive system described in Sec. V. An ansatz we present is well motivated with

- For low packing fraction, that is, $\eta \sim \dfrac{Na^3}{V} \ll 1$, the system should be well-describable by regular well-studied non-interacting case
- Jastrow factors use is justified here [11]

With this in mind, we may try to see the difference between the two cases. Unfortunately, it is merely impossible to benchmark against [1] with our low computational power (we would need to perform runs with few thousands particles, while we did few hundreds at max). For the simulation, we most of the time use $a = \dfrac{0.0043}{\sqrt{\omega}}$ and $L = 0.1$.

For all relevant results, see Tabl. I. As we see, we can roughly compare $\langle E_L \rangle / N$ fraction with left area of Fig. 2 in [1]. We note correct (i.e. growing) behaviour with starting point of ~ 2.4 (our code exactly gives 2.4). α parameter behaviour does not show significant declination from $\alpha_0 = 0.5$ – the declination does not exceed $+0.13$.

Interestingly enough, high packing fraction did not contribute too much to the result. If one takes $a = 0.043$, the packing fraction roughly becomes (N=10)

$$\eta \approx \frac{\pi N a^3}{6L^3} = 0.4, \qquad (9)$$

yet the energy extracted equals

$$E(\eta \approx 0.4, \alpha \approx 0.48) = 26.4997 \pm 0.001, \qquad (10)$$

which, even though differs notably from Tabl. I, still follows the same pattern. We leave this question for other researches.

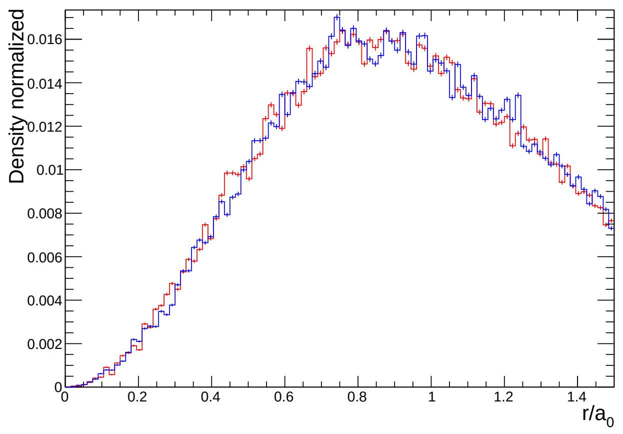

FIG. 4. Density map for non-interactive(red)/repulsive(blue) systems.

Last, but not least, given rather similar behaviour of repulsive system and non-interactive one, we compared density distribution as a function of radius. This shows an interesting observation – there is a statistically significant difference between the distributions, but there is no apparent qualitative change in the curve behaviour.

E. Numerical results with error estimation

The results for various runs are shown at Tabl. I for Metropolis (brute force) results and Tabl. II for Metropolis-Hastings (Importance sampling) results. These are all run with the cylindrical trap Hamiltonian, and with particle interactions on. For each run a total of 10^6 Monte Carlo cycles where used, split on 8 threads running in parallel, and averaging the results as described earlier.

We omit the corresponding presentation for the simpler systems, since without interaction the energy ends up the same regardless of number of particles. The agreement with exact solution, and essentially zero variance in those runs also make statistical error analysis not that relevant.

First off we note that the energy per particle is slightly higher with more particles present. This is likely due to a larger spread caused by repulsive force, that places some particles where the potential energy is higher, yet full analytic explanation may not be trivial.

The standard deviations where calculated as described in section II F. We see that for both algorithms, the standard deviation from Blocking method (resampling after the run) is about a factor of ten larger, than the one estimated during Monte Carlo simulation. This is to be expected, as the value from the simulation suffers from auto-correlation and should be too optimistic.

Furthermore we see that standard deviation increases with number of particles. It's interesting to note that both computation time and standard deviations both grows considerable with larger systems (not necessarily

N	5	10	20	50	100
$\langle E_L \rangle$	12.1287	24.3984	49.3465	127.345	266.469
time	1m16s	4m29s	15m59s	1h40m13s	$\sim$ 6h
σ_{MC}	6.100e-05	4.582e-05	0.000378	0.00198	0.00373153
σ_{BL}	0.000415	0.0005092	0.004421	0.03588	0.10089
$\langle E_L \rangle /N$	2.42574	2.43984	2.467325	2.5469	2.6647
σ_{BL}/N	0.0000830	0.0000510	0.000221	0.000718	0.00101
α	0.499227	0.497736	0.494389	0.479778	0.473841
Accept.	0.8884	0.8878	0.8899	0.8927	0.8969

TABLE I. Table showing results for MCMC runs with Metropolis "brute force" algorithm. The setup is a cylindrical trap, with various number N of particles, with 10^6 Monte Carlo cycles, and particle interaction active. The table shows energy $\langle E_L \rangle$, runtime, standard deviation from MCMC calculation σ_{MC} and by blocking method σ_{BL}, parameter α and "Accept." for acceptance rate.

N	5	10	20	50	100
$\langle E_L \rangle$	12.1285	24.3989	49.3503	127.268	266.22
time	1m8s	4m4s	16m3s	1h40m	7h30m
σ_{MC}	5.88E-05	1.49E-04	0.000363	0.001175	0.002871
σ_{BL}	0.000225	0.000607	0.001864	0.007497	0.026295
$\langle E_L \rangle /N$	2.4257	2.43989	2.467515	2.54536	2.6622
σ_{BL}/N	4.51E-05	6.07E-05	9.32E-05	0.00015	0.000263
α	0.498861	0.497425	0.495655	0.491844	0.486479
Accept.	0.9622	0.9621	0.9612	0.9586	0.9549

TABLE II. Table showing results for MCMC runs with Metropolis-Hastings algorithm using Importance sampling. The setup is a cylindrical trap, with various number N of particles, with 10^6 Monte Carlo cycles, and particle interaction active. The table shows energy $\langle E_L \rangle$, runtime, standard deviation from MCMC calculation σ_{MC} and by blocking method σ_{BL}, parameter α and "Accept." for acceptance rate.

correlated). The computation time increase can be explained by the quadratic time complexity of particle interaction calculations.

In general the Importance sampling simulation tends to take slightly longer time than brute force. This is explained by more complicated calculations for each suggested move. Run times presented here are not that exact though, as they have run on machines with varying background load.

Another interesting thing is the acceptance rate which is close to 96% for Importance sampling, but only about 89% for brute force. This supports the idea that Importance sampling is more efficient in that it converges faster and produce a larger number of distinct samples. Both ratios are fine though although Importance sampling is better.

One striking result is that the computed standard deviation is much smaller when using Importance sampling, than brute force. This is not surprising given that Importance sampling faster leads the system to a more probably state. Thus it should have measurements more clustered in a probably region than brute force has, for a given number of cycles. This tells us that even if brute force and Importance sampling on the surface give similar result, if we where to simulate many hundreds or thousands of particles, Importance sampling is really to prefer, for statistical error reasons.

The fact that the difference in result between brute force and Importance sampling, still is larger than the standard deviation of either, should however be seen as a warning sign. There could be issues with burn-in-time not accounted for and/or too few cycles. This could be further investigated by running with many more cycles.

A thing to note in retrospect is that since Blocking method script truncates input data down to powers of 2, it would be beneficial for error analysis if number of Monte Carlo cycles where generally specified in powers of 2 rather than powers of 10. As it is now, in worse case, half the collected data might be ignored during resampling. In our specific case of 10^6 cycles, and collecting one sample per hundred for resampling, less than 20% of data is truncated away during resampling.

F. Parallelization

In order to confirm the speedup by parallelization, the simulations where timed with the Linux time command. We'll take a closer look at the run on 50 particles from last section. The code starts 8 threads, hardcoded to this value because we know we are running it on laptops with 8 CPU cores.

The real runtime was 100m13s, while the so-called user-time indicating actual CPU power used was 768m8s. Monitoring CPUs also showed 100% usage of all cores most of the time. This indicates a speedup of about 7.7 - pretty close to the desired linear speedup of 8.

This is as expected since we go for an approach of trivially parallelize by running independent Monte Carlo simulations. Combining the results is a relatively simple operation.

IV. CONCLUSION

We implemented a general purpose VMC software for BEC simulation in cyllindric magnetic trap, which proves robust and, whenever applicable, corresponds to results from [1]. The optimization of trial wave function is performed for various packing fractions, with most variational parameters only slightly deviating from non-interacting case. The software is, with some level of confidence, ready to be reapplied to other quantum systems.

REFERENCES

[1] J. L. DuBois and H. R. H. R. Glyde, "Bose-Einstein condensation in trapped bosons: A variational Monte Carlo analysis," *Phys. Rev. A **63**, 023602*, vol. 63, pp. 1–17, 2001.

[2] L. P. F. Dalfovo, S. Giorgini and S. Stringari, "Theory of Bose-Einstein condensation in trapped gases," *Mod. Phys. 71, 463*, 1999.

[3] M. Hjorth-Jensen. Lecture notes fall 2015 (accessed: 31.03.2023). [Online]. Available: https://raw.githubusercontent.com/CompPhysics/ComputationalPhysics/master/doc/Lectures/lectures2015.pdf

[4] M. Jonsson, "Standard error estimation by an automated blocking method," *PHYSICAL REVIEW E 98, 043304*, 2018.

[5] P. Panasiuk and J. G. Jonsson. (2023) Github repository. [Online]. Available: https://github.com/jgjonsson/jgj-variational-monte-carlo

[6] (2023) Github repository. [Online]. Available: https://github.com/mortele/variational-monte-carlo-fys4411

[7] M. Hjorth-Jensen. Variational monte carlo methods. [Online]. Available: https://compphysics.github.io/ComputationalPhysics2/doc/LectureNotes/_build/html/vmcdmc.html#importance-sampling

[8] ——. Gradient methods. [Online]. Available: https://compphysics.github.io/ComputationalPhysics2/doc/LectureNotes/_build/html/gradientmethods.html

[9] H. G. P. H. Flyvbjerg, "Error estimates on averages of correlated data," *The Journal of Chemical Physics, vol 91, issue 1*, vol. 91, pp. 1–17, 1989.

[10] M. Hjorth-Jensen. Resampling methods: Bootstrap. [Online]. Available: https://compphysics.github.io/ComputationalPhysics2/doc/LectureNotes/_build/html/resamplingmethods.html

[11] R. Jastrow, "Many-body problem with strong forces," *Phys. Rev.*, vol. 98, pp. 1479–1484, Jun 1955. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRev.98.1479

V. BOSONS WITHIN HARMONIC OSCILLATOR

Further we consider a system of N feebly interacting nonrelativistic bosons within oscillator potential. The system is given by

$$\hat{H} = \sum_i^N \left[-\frac{\hbar^2}{2m} \triangle_i + \frac{m\omega^2}{2} \left(x_i^2 + y_i^2 + \beta^2 z_i^2 \right) \right] + \sum_{i<j}^{N\times N} V_{int}(r_{ij}), \tag{11}$$

whereas

$$V_{int}(r_{ij}) = \infty \cdot \Theta(a - r_{ij})$$

represents an on-contact (hard-core) potential with $a = d_{boson}$.

To calculate local energy we introduce promising ansatz, based on harmonic oscillator solution and Jastrow factors [11]

$$\Psi_T(\mathbf{R}) = \prod_i^N e^{-\alpha(x_i^2+y_i^2+\beta z_i^2)} \times \prod_{k<j}^{N\times N} \left(1 - \frac{a}{r_{jk}}\right) \Theta(a - r_{jk}) \equiv \prod_i^N g(\alpha, \beta, \vec{r}_i) \times \prod_{k<j}^{N\times N} f(a, r_{jk}) \tag{12}$$

We first compute the drift force given by

$$\frac{1}{2}\vec{F}_i = \frac{\vec{\nabla}_i \Psi_T}{\Psi_T} \equiv \vec{\nabla}_i \ln \Psi_T = \vec{\nabla}_i \left[\sum_j^N \ln g(\alpha, \beta, \vec{r}_j) + \sum_{j<k}^{N\times N} \ln f(a, r_{jk}) \right] = \frac{\vec{\nabla}_i g(\alpha, \beta, \vec{r}_i)}{g(\alpha, \beta, \vec{r}_i)} + \sum_{i<k} \frac{\vec{\nabla}_i f(a, r_{ik})}{f(a, r_{ik})} + \\ + \sum_{j<i} \frac{\vec{\nabla}_i f(a, r_{ji})}{f(a, r_{ji})} = \frac{\vec{\nabla}_i g(\alpha, \beta, \vec{r}_i)}{g(\alpha, \beta, \vec{r}_i)} + \sum_{j\neq i} \frac{\vec{\nabla}_i f(a, r_{ji})}{f(a, r_{ji})} \tag{13}$$

Here

$$\frac{\vec{\nabla}_i g(\alpha, \beta, \vec{r}_i)}{g(\alpha, \beta, \vec{r}_i)} = -2\alpha(x_i, y_i, \beta z_i) \tag{14}$$

and

$$\frac{\vec{\nabla}_i f(a, r_{ji})}{f(a, r_{ji})} = \vec{\nabla}_i \ln f(a, r_{ji}) = \sum_l^{N_{dim}} \vec{e}_l \frac{\partial}{\partial x_i^{(l)}} \ln f(a, r_{ji}) = \sum_l^{N_{dim}} (\ln f(a, r_{ji}))'_2 \cdot \vec{e}_l \frac{\partial}{\partial x_i^{(l)}} \sqrt{\sum_s^{N_{dim}} (x_i^{(s)} - x_j^{(s)})^2} = \\ = \sum_l^{N_{dim}} (\ln f(a, r_{ji}))'_2 \cdot \vec{e}_l \frac{x_i^{(l)} - x_j^{(l)}}{r_{ij}} = (\ln f(a, r_{ji}))'_2 \frac{\vec{r}_{ij}}{r_{ij}} = \frac{\frac{a}{r_{ij}^2}}{1 - \frac{a}{r_{ij}}} \frac{\vec{r}_{ij}}{r_{ij}} \cdot \Theta(a - r_{ij}) \tag{15}$$

General expression for the drift force is therefore

$$\vec{F}_i = -4\alpha(x_i, y_i, \beta z_i) + \sum_{j\neq i} \frac{2a}{r_{ij} - a} \frac{\vec{r}_{ij}}{r_{ij}^2} \cdot \Theta(a - r_{ij}) \tag{16}$$

Local laplasian is given by

$$\frac{\triangle_i \Psi_T}{\Psi_T} = \frac{\vec{\nabla}_i \cdot \vec{\nabla}_i \Psi_T}{\Psi_T} = \frac{\vec{\nabla}_i \cdot \frac{\vec{\nabla}_i \Psi_T}{\Psi_T} \Psi_T}{\Psi_T} = \left(\frac{\vec{\nabla}_i \Psi_T}{\Psi_T} \right)^2 + \vec{\nabla}_i \frac{\vec{\nabla}_i \Psi_T}{\Psi_T} = \frac{1}{4} \vec{F}_i^2 + \frac{1}{2} \vec{\nabla}_i \vec{F}_i \tag{17}$$

To finalize the analytic differentiation, we have to calculate

$$\frac{1}{2}\vec{\nabla}_i\vec{F}_i = \frac{1}{2}\sum_k^{N_{dim}}\frac{\partial}{\partial x_i^{(k)}}F_i^{(k)} = -2\alpha(2+\beta)+\sum_k^{N_{dim}}\frac{\partial}{\partial x_i^{(k)}}\sum_{j\neq i}\frac{a}{r_{ij}-a}\frac{\vec{r}_{ij}}{r_{ij}^2}\cdot\Theta(a-r_{ij}) = -2\alpha(2+\beta)+\sum_{j\neq i}\frac{3a}{r_{ij}-a}\frac{\Theta(a-r_{ij})}{r_{ij}^2}+$$
$$+\sum_{j\neq i}\vec{r}_{ij}\left[\frac{1}{\frac{r_{ij}^3}{a}-r_{ij}^2}\right]'\frac{\Theta(a-r_{ij})\vec{r}_{ij}}{r_{ij}} = -2\alpha(2+\beta)-\sum_{j\neq i}\frac{a^2}{r_{ij}^2(r_{ij}-a)^2}\Theta(a-r_{ij}) \quad (18)$$

Local energy is therefore given in terms of

$$\langle E_L\rangle = \frac{\sum_i^N\left[-\frac{\hbar^2}{2m}\triangle_i+\frac{m\omega^2}{2}\left(x_i^2+y_i^2+\beta^2z_i^2\right)\right]\Psi_T+\sum_{i<j}^{N\times N}V_{int}(r_{ij})\Psi_T}{\Psi_T} =$$
$$= \sum_i^N\left[-\frac{\hbar^2}{2m}\left[\frac{1}{4}\vec{F}_i^2+\frac{1}{2}\vec{\nabla}_i\vec{F}_i\right]+\frac{m\omega^2}{2}\left(x_i^2+y_i^2+\beta^2z_i^2\right)\right]+\sum_{i<j}^{N\times N}V_{int}(r_{ij}) \quad (19)$$

Note that this result is, for most part, obtained for 3D. Less dimensions may require irreversible changes, like

$$1D: -2\alpha\cdot(2+\beta)\Rightarrow -2\alpha\cdot 1$$
$$2D: (x_i, y_i, \beta z_i)\Rightarrow (x_i, y_i)$$

VI. RESCALING HARMONIC OSCILLATOR

Hamiltonian of the form Eq. 11 may be rewritten in dimensionless units by replacing

$$\hat{H}' = \frac{\hat{H}}{\hbar\omega}$$
$$\vec{r}_i' = \vec{r}_i\cdot\sqrt{\frac{m\omega}{\hbar}}\Rightarrow$$
$$\vec{\nabla}_i' = \vec{\nabla}_i\cdot\sqrt{\frac{\hbar}{m\omega}}$$

That way,

$$\hat{H}' = \sum_i^N\left[-\frac{\hbar}{2m\omega}\triangle_i+\frac{m\omega}{2\hbar}\left(x_i^2+y_i^2+\beta^2z_i^2\right)\right]+\sum_{i<j}^{N\times N}V_{int}(r_{ij}) = \sum_i^N\left[-\frac{1}{2}\triangle_i'+\frac{1}{2}\left(x_i^{2\prime}+y_i^{2\prime}+\beta^2z_i^{2\prime}\right)\right]+\sum_{i<j}^{N\times N}V_{int}(r_{ij}) \quad (20)$$

VII. OPTIMIZER BENCHMARK

Here we present some sample runs of our optimizer. The testing was performed for 10 particles in 3D for both non-interacting and repulsive cases with $\beta = 1.0 : 2.82843$ respectively. Column printout style : $\alpha : \langle E(\alpha)\rangle : \sigma_E : \frac{\partial\langle E\rangle}{\partial\alpha}$

```
./bin/Project-1/Problem-1d/probe_optimizer_gaussian.out 3 10 100000 0.45
Iteration 0
Predictions: 0.45 15.2718 0.0152635 -4.90473
Iteration 1
Predictions: 0.55 14.9244 0.0128599 3.15006
Iteration 2
Predictions: 0.485775 15.0556 0.00408645 -1.19087
```

```
Iteration 3
Predictions: 0.510055 14.9693 0.0028061 0.7753
Iteration 4
Predictions: 0.494248 15.0207 0.00163545 -0.467697
Iteration 5
Predictions: 0.503784 14.9876 0.00106368 0.297895
Iteration 6
Predictions: 0.49771 15.008 0.000648397 -0.184017
Iteration 7
Predictions: 0.501462 14.9951 0.000412097 0.115997
Iteration 8
Predictions: 0.499097 15.0031 0.000255297 -0.0722338
Iteration 9
Predictions: 0.50057 14.9981 0.000160742 0.0453339
Iteration 10
Predictions: 0.499645 15.0012 0.000100194 -0.0283149
Iteration 11
Predictions: 0.500223 14.9992 6.28496e-05 0.0177389
Iteration 12
Predictions: 0.499861 15.0005 3.92689e-05 -0.0110922
Iteration 13
Predictions: 0.500087 14.9997 2.45963e-05 0.00694422
Iteration 14
Predictions: 0.499946 15.0002 1.53823e-05 -0.00434418
Parameters converged after 14 iterations.

  -- System info --
 Number of particles : 10
 Number of dimensions : 3
 Number of Metropolis steps run : 10^5
 Step length used : 0.1
 Ratio of accepted steps: 0.98575

  -- Wave function parameters --
 Number of parameters : 1
 Parameter 1 : 0.500034

  -- Results --
 Energy : 15
 Standard deviation Energy : 8.40683e-07
 Computed gradient : 0.00207225
```

```
    ./bin/Project-1/Problem-1d/probe_optimizer_repulsive.out 3 10 100000 0.35
Iteration 0
Predictions: 0.35 25.4792 0.0676614 -23.7958
Iteration 1
Predictions: 0.45 24.3697 0.0184879 -4.87902
Iteration 2
Predictions: 0.470504 24.4323 0.010386 -2.61566
Iteration 3
Predictions: 0.481496 24.3735 0.00580874 -1.35042
Iteration 4
Predictions: 0.487171 24.3984 0.00371874 -0.784912
Iteration 5
Predictions: 0.490469 24.4115 0.00259582 -0.453207
Iteration 6
Predictions: 0.492374 24.4186 0.00204427 -0.260673
Iteration 7
Predictions: 0.493469 24.4226 0.00179517 -0.149598
Iteration 8
Predictions: 0.494098 24.4248 0.00168674 -0.0857424
Iteration 9
Predictions: 0.494458 24.4261 0.00163867 -0.0491073
```

```
Iteration 10
Predictions: 0.494665 24.4268 0.00161626 -0.0281133
Iteration 11
Predictions: 0.494783 24.4272 0.0016052 -0.0160906
Parameters converged after 11 iterations.

  -- System info --
 Number of particles : 10
 Number of dimensions : 3
 Number of Metropolis steps run : 10^5
 Step length used : 0.1
 Ratio of accepted steps: 0.88968

  -- Wave function parameters --
 Number of parameters : 1
 Parameter 1 : 0.494851

  -- Results --
 Energy : 24.3947
 Standard deviation Energy : 0.000235303
 Computed gradient : -0.233892
```