UNION University School of Computing

Marina Radulaški

Numerical Simulations of Rotating Bose-Einstein Condensates

diploma thesis

Belgrade, 2009

Motivated by the highschool research in Petnica Science Center, done in cooperation with the Institute of Physics in Belgrade, I chose the Union University School of Computing, on my quest to gain further knowledge in the field of scientific computing. Today, after several years, I am grateful to all the three institutions for the offered education and practical skills which allowed me to complete the research presented in this thesis, conducted in Scientific Computing Laboratory, Institute of Physics, Belgrade.

Special thanks go to my supervisors, doc. Dr. Antun Balaž, who introduced me to the secrets of research, got me interested in Bose-Einstein condensation and directed my work through numerous constructive advices, as well as to doc. Dr. Radomir Janković who taught me about the modeling and simulation techniques.

I would also like to thank my family and friends for being here for me and supporting me throughout my studies.

in Belgrade, December 2009 *Marina Radulaški*

Contents

1

Introduction

Within the traditional research approach, completely parallel with analytic and synthetic approach to science, two main epistemological paradigms can be distinguished: experiment and theory. Rapid computer development enabled the appearance of the third paradigm: numerical simulations.

Motivations for numerical approach to the description of nature are various:
- the system considered is too complicated for full theoretical or numerical treatment (too many variables to consider, too many structural elements),
- microscopic behavior of the system is not completely known (effective interactions of structural elements),
- one is interested in analysis of a simplified model (in less dimensions),
- description of the original system is non-realistic in the used mathematical formalism (change to the imaginary time scale).

This approach allows for numerical experiments with systems whose behavior is (relatively) easy to implement in numerical algorithms, as well as for the investigation of theoretical models of complex natural phenomena.

This thesis studies numerical simulations of a specific physical system: ultra-cold atom gas in a rotating magneto-optical trap. At low enough temperatures, this system exhibits a phase transition – Bose-Einstein condensation. The thesis presents how to investigate Bose-Einstein condensate properties using numerical simulations. The focus is on the examination of the structure and complexity of the algorithms used in these simulations.

1.1 Bose-Einstein Condensation

In the second half of 19^{th} century, Maxwell[1] and Boltzmann[2] developed a theory which describes how particles of matter occupy energy levels in the state of thermodynamic equilibrium. In 1920, Bose[3] wrote an article in which he showed that particles of light, photons, do not obey Maxwell-Boltzmann statistics. At first, the validity of Bose's work was not recognized in Europe. This made him write in 1924 directly to Einstein[4] who translated the paper into German and got it published for Bose. Einstein not only recognized the

[1] James Clerk Maxwell (1831-1879), Scottish teoretical physicist and mathematician.
[2] Ludwig Eduard Boltzmann (1844-1906), Austrian physicist.
[3] Satyendra Nath Bose (1895-1974), Indian physicist.
[4] Albert Einstein (1879-1955), German theoretical physicist.

significance of Bose's work, but also expanded his conclusion to mass particles with an integer value of spin[5] value. The theoretical work of these two physicists predicted a new phase for bosons[6] at low temperatures (close to the absolute zero). Today, that phase is known as Bose-Einstein condensate (BEC), while the according phase transition is called Bose-Einstein condensation.

In order to gain a better understanding of the underlying phenomenon, we will start from the classification of elementary particles into bosons and fermions. Bosons (e.g. photons) have integer spin values, while fermions (e.g. electrons) half-integer. More than one boson can occupy the same quantum state characterized by the equal values of all quantum numbers, while fermions are forbidden to do so by the Pauli[7] exclusion principle. This is nicely illustrated through the examples: on the one hand, laser light consists of photons, all in the same quantum state, while on the other hand, during the filling of atomic orbitals, only two electrons can occupy the same orbital and must have opposite spin values to fulfill Pauli principle.

Bose-Einstein condensation [1, 2, 3] represents the macroscopic occupation of the ground (lowest) energy state of a boson system. At low temperatures this occurs naturally since every physical system tends to minimize its energy. Condensate particles are in a macroscopic coherent state. Therefore, BEC is the origin of the superfluidity and superconductivity phenomena, where particles travel with no internal resistance (viscosity, electrical resistance).

Fermi[8] thought that BEC can't be achieved in practice. Theoretically, this phase transition happens when the interaction between the particles is small. Fermi believed that interactions in nature are too high to allow the transition. Today we know Fermi was not right and this discovery started a whole new field in physics (cold quantum gases).

The development of cooling techniques achieving the temperatures of the order of 100nK was crucial for the experimental realization of BEC. After decades of work, in 1995, BEC was finally observed in laboratory conditions [4, 5]. The Nobel prize in physics was awarded for this discovery in 2001. BEC is most often realized by cooling neutral alkali atoms (isotopes with an even number of neutrons in the nucleus, which act as bosons), such as ^{87}Rb, ^{23}Na, ^{7}Li, ^{133}Cs, to temperatures close to the absolute zero. When the temperature of a boson system gets below the critical temperature of Bose-Einstein condensation, particles start to macroscopically occupy the lowest energy state.

This thesis considers the case of an ideal boson gas in a grand canonical ensemble [6], which is the system of noninteracting bosons that allows for the energy and particle exchange with its surroundings.

The main physical variable for the BEC phenomenon is the condensation temperature T_c. It represents the temperature below which the number of particles N_0 occupying the ground state is macroscopic, and above which a negligible number of atoms can be found in the ground state. For a system of ideal bosons [1, 2, 3], the number of atoms in the ground

[5] Spin represents one of the main parameters of a particle quantum state description.
[6] Bosons are the particles whose spins have integer values.
[7] Wolfgang Ernst Pauli (1900-1958), Austrian theoretical physicist.
[8] Enrico Fermi (1901-1954), Italian physicist.

state can be determined by the equation:

$$N_0 = N - \sum_{m=1}^{\infty} [e^{m\beta E_0} Z_1(m\beta) - 1],$$

where N represents the total number of particles in the system (number of atoms in a magneto-optical trap), $\beta = \frac{1}{k_B T}$ is called the inverse temperature, k_B is the Boltzmann constant, and T is the thermodynamic temperature. In the upper equation E_0 represents the ground state energy for a single-particle potential, and $Z_1(\beta)$ is a single-particle partition function, defined as:

$$Z_1(\beta) = \sum_{k=0}^{\infty} e^{-\beta E_k}.$$

To calculate the condensation temperature, the energy spectrum of the theory is required. In other words, one needs to calculate the values of the single-particle system energy levels E_k. These are the eigenvalues of the system energy operator $\hat{H}$ (Hamiltonian). We used the method described in Refs. [7, 8] to determine the spectrum of the theory. The system considered evolves in a short (imaginary) time ϵ and the transition amplitudes from the position $\mathbf{r}$ to the position $\mathbf{r}'$ for the given time have the form $A(\mathbf{r}, \mathbf{r}'; \epsilon) = \langle \mathbf{r}' | e^{-\epsilon \hat{H}} | \mathbf{r} \rangle$. The eigenvalues of the evolution operator matrix A defined this way are given as $e^{-\epsilon E_k}$. Therefore, if matrix A is known and there is a way to calculate its eigenvalues, it is also possible to calculate energy levels E_k for the system of ideal bosons, as well as BEC condensation temperature and other global thermodynamic properties of the condensate.

The matrix elements of the evolution operator can not be calculated in an exact way for the general case and this represents the main problem in non-trivial potential system examination. On the contrary, the transition amplitudes $A(\mathbf{r}, \mathbf{r}'; \epsilon)$ can be numerically efficiently calculated by the effective action method, developed in Refs. [7, 8, 9, 10, 11]. The error introduced to the energy level values by this method scales with ϵ^p, where integer p represents the chosen effective action level (see 3.1 in Results). Thus, for short evolution times ($\epsilon < 1$), the error of the calculation of the transition amplitude becomes negligible for big enough p. The elements of the matrix A can be calculated by choosing the appropriate space discretization of the system [7, 8, 9, 10, 11]. The eigenvalues and eigenvectors can be numerically found in an exact way by using one of the diagonalization[9] algorithms. This leads to the solution of the examined problem.

1.2 Rotating Bose-Einstein Condensate

Condensates in rapidly rotating magneto-optical traps take important part in the current BEC experiments. Their significance lies in the contribution to the understanding of the formation and evolution of quantum vortices, an intriguing fundamental physics phenomenon [12, 13], which represents the reaction of the superfluid to the excitations of the rotational

[9] Diagonalization represents transformation of a matrix into its equivalent diagonal form; when the matrix is transformed into this form its nonzero elements (found on the main diagonal) are equal to the eigenvalues of the matrix.

system - quantum rotations. In the experiment, this can be seen as a formation of smaller independent whirlpools, which are characterized by the vorticity – a variable that takes quantized values.

In the experiment of Dalibard and coworkers [14], atoms of ^{87}Rb are trapped in harmonic potential with a small anharmonic part in $x-y$ plane. The whole system rotates around z-axis with the angular speed Ω, which makes atoms feel the potential:

$$V(x,y,z) = \frac{M}{2}\omega_\perp^2(1-r^2)(x^2+y^2) + \frac{M}{2}\omega_z^2 + \frac{k_n}{24}(x^2+y^2)^2,$$

where M represents the mass of a particle, k_n the anharmonicity, $\omega_\perp = 407.15$Hz the harmonic frequency of the trap in $x-y$ plane, $\omega_z = 69.115$Hz the frequency of the trap along the z-axis, and r is the parameter of the system rotation, $r = \frac{\Omega}{\omega_\perp}$. In order to operate with dimensionless values, natural units were introduced where $M = \hbar = 1$. The accordingly rescaled anharmonicity k_n is expressed in the units of $\frac{\hbar}{M^2\omega_\perp^3}$ and equals $k_n = 0.00195$, while the length is expressed in the units of $\sqrt{\frac{\hbar}{M\omega_\perp}}$.

The potential appears in the calculation of the transition amplitudes which is the first step in the determination of the condensation temperature within the considered approach.

1.3 Purpose of the Thesis

In the determination of the BEC condensation temperature using the described approach, the main numerical challenge is given by the calculation of the eigenvalues and eigenvectors of the evolution operator. Numerically, this problem can be solved by the diagonalization of the matrix of spatially discretized operator. Accurate description of such a physical system demands a big number of eigenvalues and eigenvectors, which is possible only with a very fine discretization. Surely, this requires matrices of big dimensions which makes diagonalization very demanding in the means of processor time and memory.

There are many availible algorithms for diagonalization. The goal of this thesis is to investigate the accuracy, time and memory complexity of the Lanczos algorithm [15] compared to the standard implementation of the full diagonalization algorithm, for solving the eigenproblem of the evolution operator for the described rotating Bose-Einstein condensate, considered as a gas of ideal ^{87}Rb bosons.

2

Numerical Simulations of Bose-Einstein Condensates

Within the approach used for the numerical description of Bose-Einstein condensates, it is necessary to introduce an appropriate space discretization. The discretization has to be chosen in a controlled way, so that the error can be estimated and therefore avoided for some set of parameters. In Ref. [7], the analytical error introduced by the discretization is approximated, while in Ref. [8], it is shown how this error can be eliminated for the appropriately chosen set of discretization parameters. This paper uses the mentioned approach. The spatial coordinates of the atoms are chosen to take values $n\Delta$, where the integer n takes $L = 2N + 1$ values from the set $[-N, N]$. Because numbers are represented in binary system, many numerical algorithms run faster for matrices with even size. Therefore, the set of values for n is usually redefined to $n \in [-N, N)$, which makes the number of points per spatial dimension even, $L = 2N$. With this kind of discretization, the matrix of the evolution operator is represented as a $D \times D$ matrix, where $D = L^d$, and d is the number of dimensions. The most important problem in numerical simulations of Bose-Einstein condensates is to determine the eigenvalues λ of this matrix:

$$Av = \lambda v,$$

where v represents the according eigenvector.

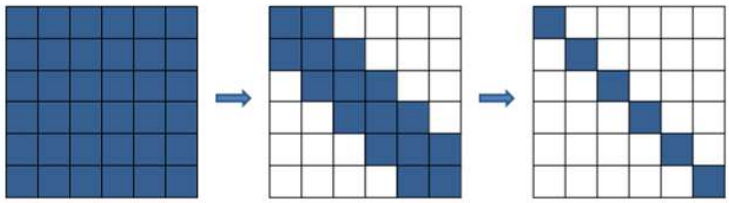
Since the examined matrix A is real $A \in \mathbb{R}^{D^2}$ and symmetric $A^T = A$, there exists an orthogonal[10] matrix P which can be transformed into a diagonal matrix by the similarity operation $P^T AP$. Diagonal values of $P^T AP$ are eigenvalues of the matrix A, while the vectors that matrix P consists of (its columns) are the eigenvectors of the matrix A. The number of eigenvalues and eigenvectors obtained in such a way is equal to the linear matrix dimension D.

The following sections present the two diagonalization algorithms investigated in this thesis: full diagonalization algorithm and Lanczos algorithm.

[10] Orthogonal matrix P is characterized by the property $PP^T = I$.

2.1 Full Diagonalization Algorithm

Full diagonalization algorithm determines all eigenvalues and eigenvectors exactly. Their number is equal to the matrix linear dimension D. The algorithm consists of two steps illustrated on Figure 2.1.



2.1: The illustration of the full diagonalization algorithm: matrix is firstly transformed to its tridiagonal form and then to diagonal.

In the first step, the matrix is being transformed into a real symmetric tridiagonal[11] matrix

$$A = QA_QQ^T$$

where Q is a suitably chosen orthogonal matrix. To reach this form, the DR decomposition is used. It was developed in 1961 by Francis[12] and Kublanovskaya[13].

In the second step, eigenvalues and eigenvectors of the tridiagonal matrix A_Q are being calculated. This matrix can be decomposed as $A_Q = SA_DS^T$, where A_D is a diagonal matrix whose elements are at the same time its eigenvectors, while matrix S is made of eigenvectors of the matrix A_Q. If Z is set to be $Z = QS$ then the relation $A = ZA_DZ^T$ holds. Therefore the values on the diagonal of A_D represent the eigenvalues of the matrix A and Z is the matrix of the eigenvectors of the matrix A. One can notice that the eigenvalues for the matrices A, A_Q and A_D are equal.

The accuracy of this diagonalization algorithm depends solely on the computer numerical precision. The algorithm itself does not introduce an error in the calculation of the matrix eigenvalues, but its numerical implementation limits the number of significant digits obtained. Since the calculated eigenvalues in the problem examined are equal to $e^{-\epsilon E_k}$, a small change in the value of the energy E_k causes exponential change of the eigenvalue, which illustrates the importance of the accurate calculation within the investigated problem. This thesis uses LAPACK [17] implementation of the full diagonalization algorithm.

2.2 Lanczos Diagonalization Algorithm

Lanczos algorithm [15] is an exact iterative algorithm for calculation of an a itrary number (maximum D) of matrix eigenvectors and eigenvalues. It is especially suitable for large

[11] Tridiagonal matrix has all the elements equal to zero accept those on the main diagonal and the first two parallel diagonals.

[12] John G. F. Francis (1934-), English computer scientist.

[13] Vera Nikolaevna Kublanovskaya (1920-), Russian mathematician.

sparse[14] matrices. It was developed by Lanczos[15] in the middle of the 20^{th} century. Like the full diagonalization algorithm, it consists of two steps illustrated on Figure 2.1: transformation into tridiagonal form T and diagonalization of the tridiagonal matrix.

Lanczos algorithm is based on the following. Let q_1 be a random vector from which an array of vectors q_k is formed by multiplication with a matrix A as $q_{k+1} = Aq_k$. It can be shown that for such an array the value $\frac{\|q_{k+1}\|}{\|q_k\|}$ will converge to the largest eigenvalue and $\frac{q_k}{\|q_k\|}$ to the corresponding eigenvector of the matrix A. Vectors q_k form a so-called Krylov[16] subspace. Lanczos algorithm uses these vectors to determine the elements α_i, β_i of the tridiagonal form T of the matrix A:

$$\begin{pmatrix} \alpha_1 & \beta_1 & 0 & . & . & . & 0 \\ \beta_1 & \alpha_2 & \beta_2 & . & . & . & 0 \\ 0 & \beta_2 & \alpha_3 & . & . & . & 0 \\ . & . & & . & & & . \\ . & . & & & . & & . \\ . & . & & & & . & \beta_{k-1} \\ 0 & 0 & 0 & . & . & \beta_{k-1} & \alpha_k \end{pmatrix}$$

The number of iterations depends on the number of desired eigenvalues and the convergence of the algorithm. Typically, the number of iterations is twice the number of desired eigenvalues n_{eig}. In order to improve the accuracy of the algorithm, it is necessary to orthogonalize the set of vectors q_k after every iteration.

The iterations of the Lanczos algorithm can be presented as a pseudo-code:

$q_1 = \frac{q_{rand}}{\|q_{rand}\|}$

for $k = 1$ to $m - 1$ do
  $q_{k+1} = Aq_k$
  $\alpha_k = q_k^T q_k$
  $q_{k+1} = q_{k+1} - \alpha_k q_k$

  if $k > 1$ then
    $q_{k+1} = q_{k+1} - \beta_{k-1} q_{k-1}$
  end if

  reorthogonalize
  $\beta_k = \|q_{k+1}\|$

  if $\beta_k = 0$ then
    $q_{k+1} = \frac{q_{rand}^{new}}{\|q_{rand}^{new}\|}$
    reorthogonalize
  end if

[14] Matrices with a large number of zero elements.
[15] Cornelius Lanczos (1893-1974), Hungarian mathematician and physicist.
[16] Alexei Nikolaevich Krylov (1863-1945), Russian navy engineer and applied mathematician.

$q_{k+1} = \frac{q_{k+1}}{\beta_k}$
end for

Function *reorthogonalize* orthogonalizes the set of vectors q_k using the standard Gram[17]-Schmidt[18] procedure [16].

The advantage of Lanczos algorithm compared to full diagonalization algorithm is that the matrix does not change through the iterations, therefore it is not necessary to store it in the memory. When a function for calculating matrix elements exists, it can be used as an alternative to the storage of the matrix. Such a function is then called in every iteration to calculate the elements needed. This allows for the diagonalization of very large matrices which are too big to be stored in computer memory.

Of course, the storage of the matrix in the memory or the calculation of the matrix elements through the iterations, influence the time and memory needed for the execution of the program.

The accuracy of the Lanczos algorithm depends not only on numerical precision, but also on the size of the evolution operator matrix, as well as on the desired number of eigenvalues. The algorithm does not converge for all sets of parameters and performs best for problems where a small number of eigenvalues is required, $n_{eig} \lesssim 100$. Despite these flaws, fast convergence and the possibility to work with large matrices favor Lanczos algorithm in many applications.

2.3 Implementation

All algorithms described have been implemented in C/C++ programing language.

For full diagonalization (FD) LAPACK [17] library was used. It is a library for the solving of linear algebra problems, written in Fortran 77 programing language. The corresponding code is given in the Appendix A.1. The time complexity of this algorithm is

$$\tau^{FD} = O(D^3),$$

where D represents the linear dimension of the matrix being diagonalized. Memory complexity of the algorithm is

$$M^{FD} = O(D^2),$$

and the main factors of this dependence are due to the need to store the matrix which is being diagonalized, as well as the calculated eigenvectors.

The program for Lanczos diagonalization has been developed in the C/C++ programing language and is given in the Appendix A.2. The program has two main modes which have been studied separately:

1. Evolution matrix is being calculated at the beginning of the algorithm and stored

[17] Jørgen Pedersen Gram (1850-1916), Dutch actuary and mathematician.
[18] Erhard Schmidt (1876-1959), German mathematician.

completely in the memory during the algorithm execution (LDm).

2. Evolution matrix is not stored in memory, but its elements are being calculated during the algorithm iterations when needed ($LDnm$).

For both implementations of Lanczos algorithm the time complexity is approximated to

$$\tau^{LD} = O(n_{eig}^3 + n_{eig}D^2) \approx O(n_{eig}D^2),$$

but in the second case ($LDnm$) the prefactor is significantly higher and depends on the complexity of transition amplitude function, or in this case the level of effective action p. In the Appendix A.3, an example of the effective action function is shown for one-dimensional system and the level $p = 6$. The memory complexity for the implementation with the storage of the matrix is approximated as

$$M^{LDm} = O(D^2).$$

Here, the memory is dominated by the matrix being diagonalized, while in the case without the storage of the matrix the required memory is much smaller and the main part of the memory is used for the storage of the vectors q_k which makes the memory complexity much lower

$$M^{LDnm} = O(n_{eig}^2 + n_{eig}D) \approx O(n_{eig}D).$$

Besides the mentioned theoretical approximations of the algorithms complexities, all these dependencies are shown in the next chapter based on the results of the numerical simulations where the runtime and the occupied memory were measured.

3

Results

The results are obtained from the simulations run in Linux on computers with quad-core 64-bit Intel Xeon E5345 processors and dual-core 64-bit Intel Core 2 Duo T5500 with 32-bit OS. All the equations have been rescaled in order to operate with dimensionless units.

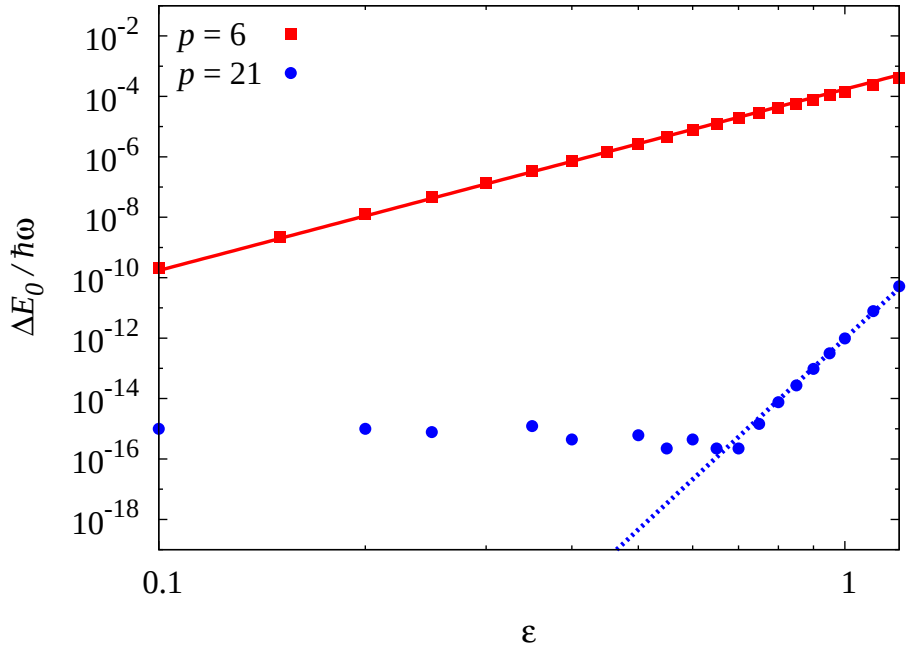
The first results shown present the errors introduced by the used approach: error due to the approximation of matrix elements, error and numerical error. Afterwards, the time and memory complexities are analyzed and, at the end, the global and local properties of the ideal ^{87}Rb gas in a rotating magneto-optical trap in BEC phase are shown.

In order to check the accuracy of the effective action method used for the calculation of the evolution matrix elements and determine the time interval of the propagation ϵ for which the satisfactory level of precision is obtained, the values of harmonic oscillator energy levels calculated by this method are compared to the known analytic results. Figure 3.1 shows the error introduced into the ground state energy by the effective action method which has a power-law dependence on propagation time and is proportional to ϵ^p, where p is a chosen effective action level. For the level $p = 21$, on the left side of the plot, computer numerical precision limits the precision of the result more than applied approximative method, and the error saturates around 10^{-15}.

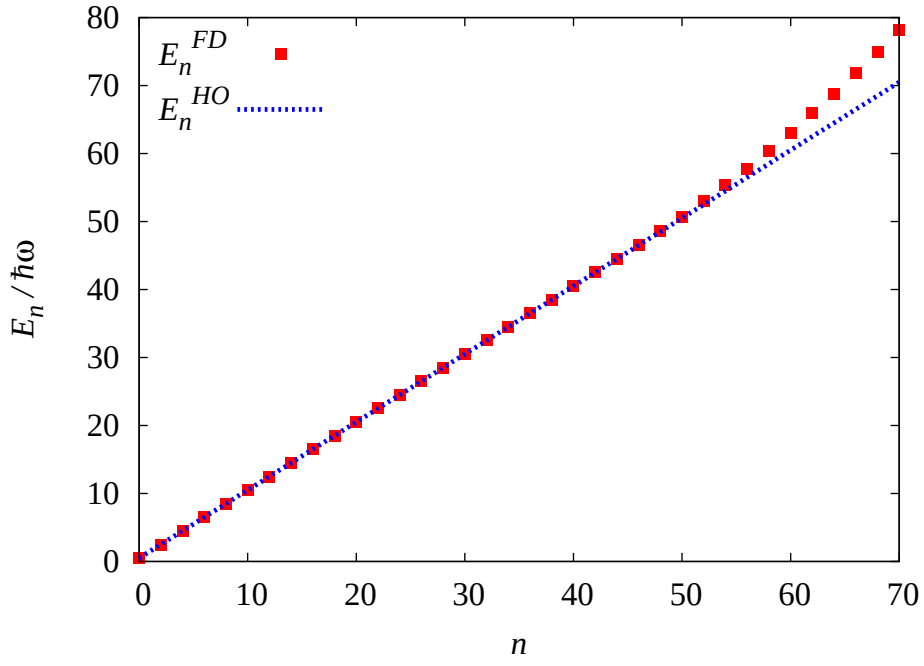
Due to the introduced space discretization and the numerical error in the calculation of matrix A, the eigenvalues and eigenvectors for only a certain set of the eigenstates can be determined precisely. Figures 3.2 and 3.3 analyze the correctness of the energy level evaluation (eigenvalues), while Figures 3.4 and 3.5 show the correctness of the state vectors (eigenvectors). These plots illustrate a similar accuracy of the full diagonalization and the Lanczos algorithm.

In the numerical approach to the given problem, the matrix diagonalization represents the most expensive step in the use of computer resources. Figure 3.6 shows that time complexity of the full diagonalization algorithm scales as the third power of the matrix dimension.

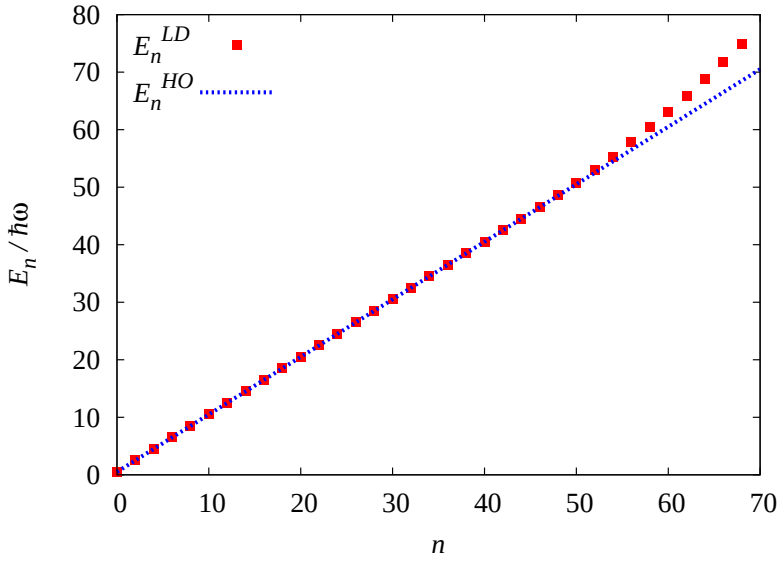
Figures 3.7, 3.8, 3.9 show that the time complexity for all implementations of Lanczos diagonalization algorithm (with and without the storage of the matrix which is being diagonalized, for different number of desired eigenvalues and eigenvectors, for different complexity of evolution operator functions, which corresponds to different levels of effective action p) scales as the square of matrix dimension D^2, while the multiplicative constant of D^2 rises



3.1: The error of calculation of the ground energy state for one-dimensional harmonic oscillator ($E_0 = \frac{1}{2}\hbar\omega$) using effective action method with levels $p = 6$ and $p = 21$ as a function of propagation time ϵ. Fitted power law dependences $\Delta E_0^{p=6}(\epsilon)/\hbar\omega = 1.730(9)\cdot 10^{-4}\ \epsilon^6$ and $\Delta E_0^{p=21}(\epsilon)/\hbar\omega = 9.67(9)\cdot 10^{-13}\ \epsilon^{21}$ are also shown. Simulation used matrix of dimension $D = 2000$ for the area $|x| \leq 10$, $\Delta = 0.01$ and rotation parameter $r = 0$.



3.2: The first 70 energies of one-dimensional harmonic oscillator calculated with full diagonalization algorithm (FD) and analytical solution (HO) given with $E_n = (n + \frac{1}{2})\hbar\omega$. Simulation used matrix of dimension $D = 2000$ for the area $|x| \leq 10$, $\Delta = 0.01$, rotation parameter $r = 0$ and the level of effective action $p = 21$.



3.3: The first 70 energies of one-dimensional harmonic oscillator calculated with Lanczos algorithm (LD) and analytical solution (HO) given with $E_n = (n + \frac{1}{2})\hbar\omega$. Simulation used matrix of dimension $D = 2000$ for the area $|x| \leq 10$, $\Delta = 0.01$, rotation parameter $r = 0$ and the level of effective action $p = 21$.

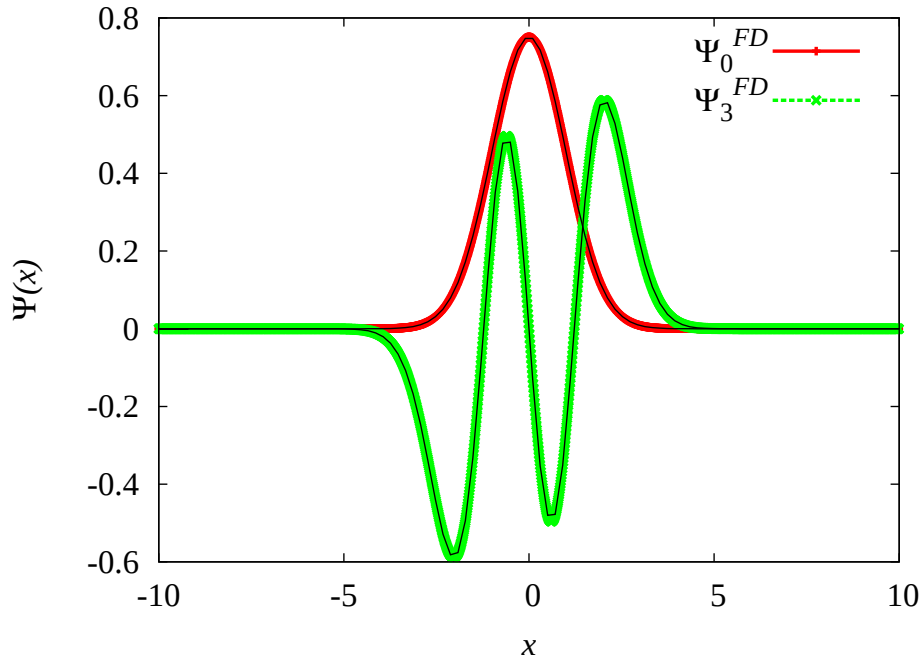
with the number of desired eigenvalues and complexity of the calculation of the matrix elements of the evolution operator.

Figures 3.10 and 3.11 show that, for both implementations of Lanczos algorithm, the time complexity is linear in the number of the desired eigenvalues. Figures 3.7–3.11 verify that the time complexity of Lanczos algorithm is $O(n_{eig}D^2)$, as it was stated in the previous chapter.

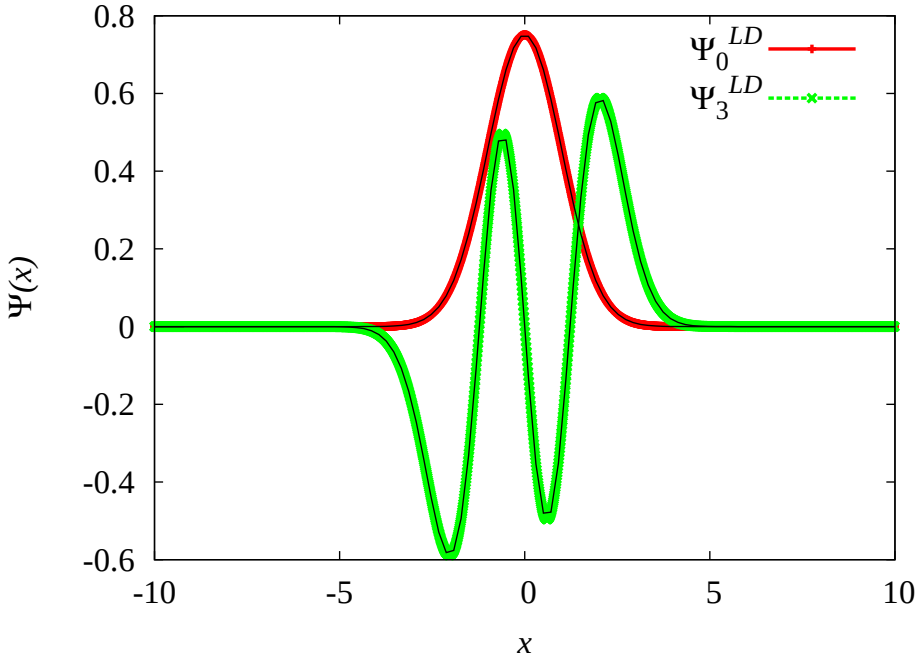
Numerical diagonalization is also demanding memory-wise. Figure 3.12 shows that for full diagonalization the memory complexity scales as a square of the matrix linear dimension D. The same memory complexity is obtained on Figure 3.13 for Lanczos algorithm with the storage of the matrix, while a significant improvement in memory usage is achieved with Lanczos algorithm without the storage of the matrix, which memory usage is linear in the matrix dimension D, as it is shown on Figure 3.14.

The developed algorithm of Lanczos diagonalization of the evolution operator matrix A has been applied to investigate the ideal boson gas of ^{87}Rb trapped in a rotating magneto-optical trap. Figure 3.15 shows the occupancy of the ground state for temperatures less than or equal to condensation temperature T_c, for which the ground state occupancy becomes practically zero. This dependence is well known in the literature [1, 2, 3] and can be used to determine T_c. Figure 3.16 shows how the condensation temperature rises with the number of bosons in the system. Theoretically [1, 2, 3], in the case of harmonic oscillator ($k_n = 0$), this dependence has the form $T_c \sim N^{1/2}$ for a condensate in a harmonic trap. This dependence is similar for the studied system with small anharmonicity ($k_n = 0.00195$), which is shown through an appropriate power-law fit. Figure 3.17 shows particle density for the position in the $x - y$ plane at temperature below T_c, close to the absolute zero. The obtained sharp peak has the characteristic shape which is seen in BEC experiments [1, 2, 3]

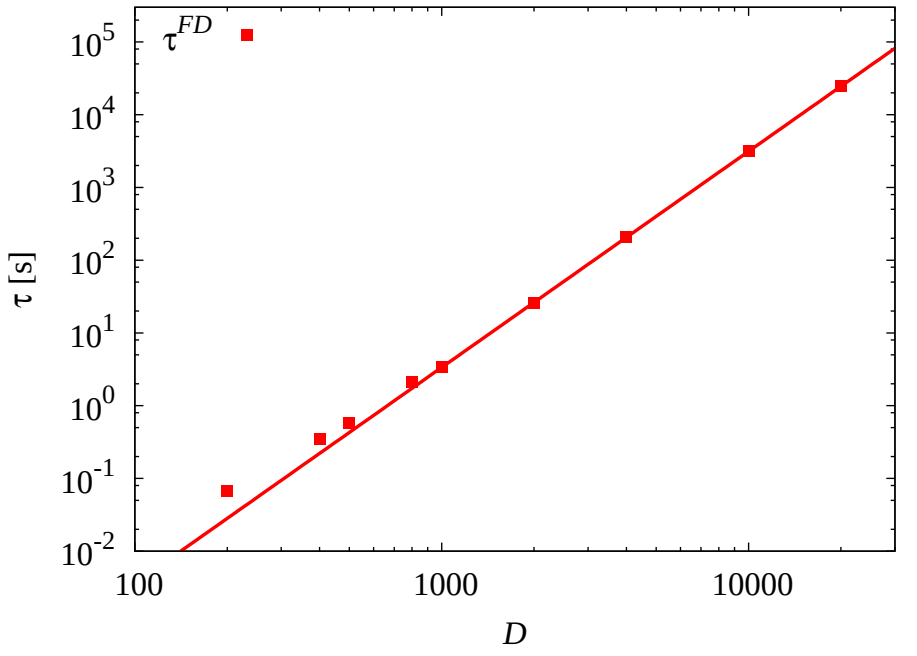
and represents a confirmation for the system phase transition.



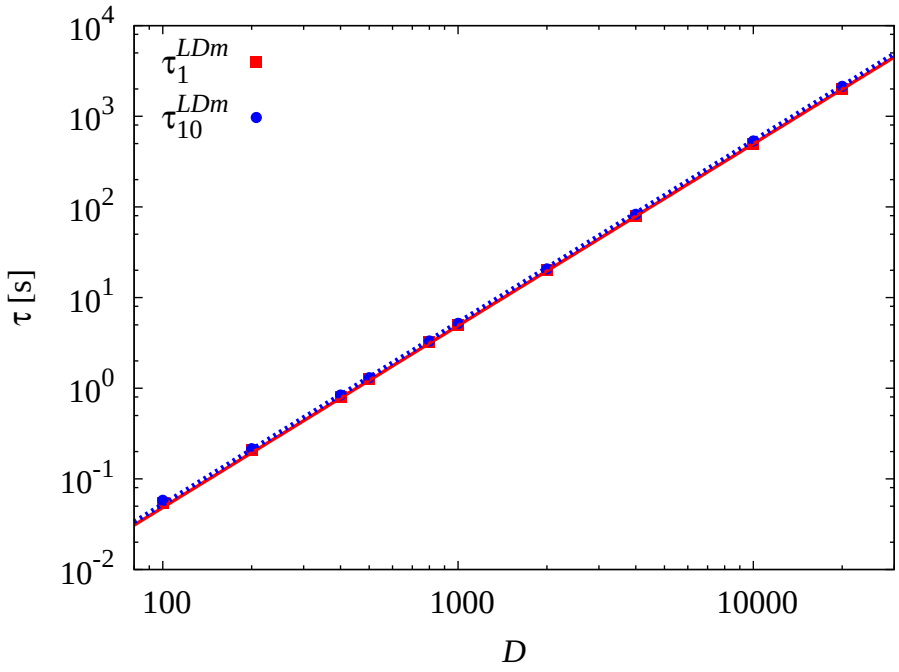
3.4: The eigenstates of one-dimensional harmonic oscillator for ground and third energy level calculated with full diagonalization algorithm (FD). Lines show analytic solutions $\Psi_n(x) = \sqrt{\frac{1}{2^n n!}}\pi^{-\frac{1}{4}} e^{-\frac{x^2}{2}} H_n(x)$, where H_n represents the Hermite polynomial. Simulation used matrix of dimension $D = 2000$ for the area $|x| \leq 10$, $\Delta = 0.01$, rotation parameter $r = 0$ and the level of effective action $p = 21$.



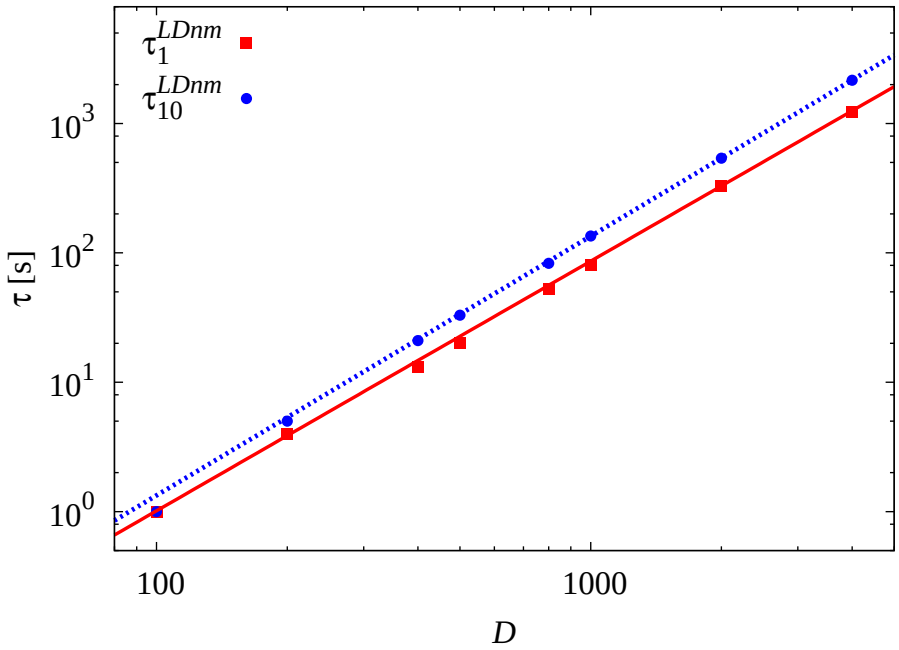
3.5: The eigenstates of one-dimensional harmonic oscillator for ground and third energy level calculated with Lanczos algorithm (LD). Analytic solution and the rest of parameters are same as in the previous figure.



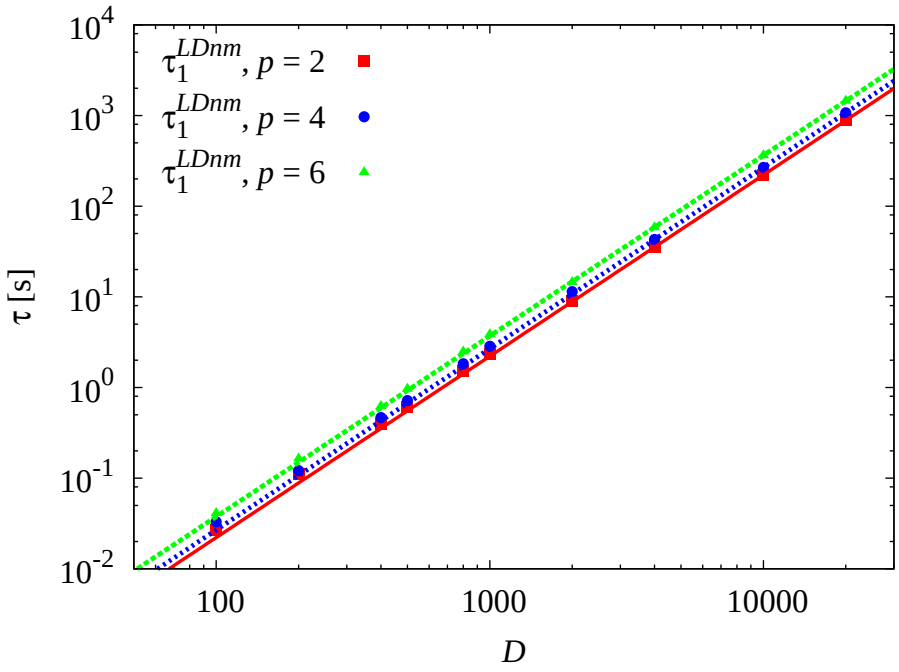
3.6: Full diagonalization (FD) runtime dependence on the size of the matrix being diagonalized fitted by a power-law $\tau^{FD}(D) = 4.13(2) \cdot 10^{-9}\mathrm{s} \cdot D^{2.9704(5)}$. One-dimensional BEC potential with anharmonicity $k_n = 10^{-3}$ has been diagonalized for the area $|x| \leq 10$, $\Delta = 0.01$, rotation parameter $r = 0$ and the level of effective action $p = 21$.



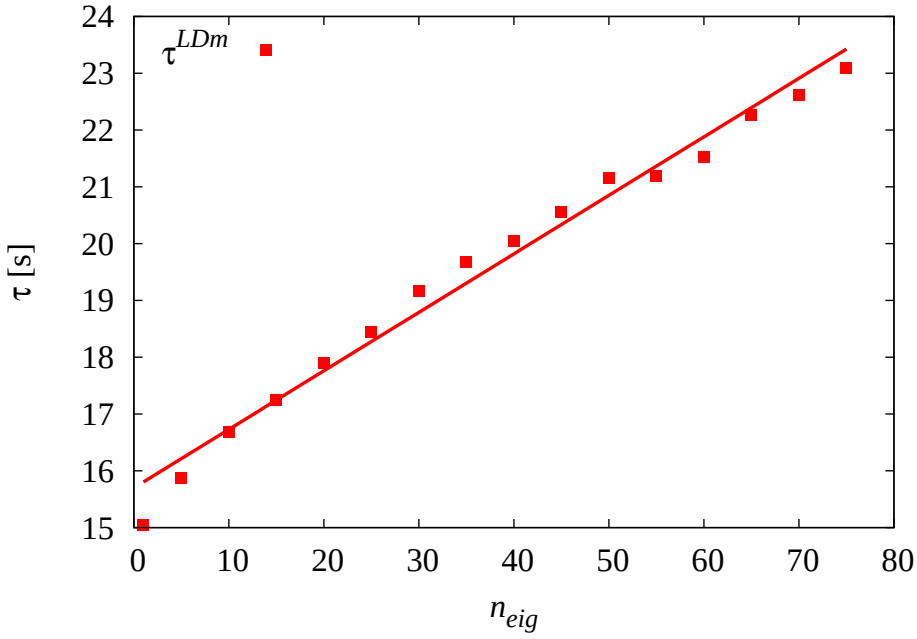
3.7: The runtime of Lanczos diagonalization (LDm) algorithm with the storage of the matrix while calculating 1 and 10 eigenvalues as a function of the matrix size, fitted by power-laws $\tau^{LDm}_{n_{eig}=1}(D) = 4.7(3) \cdot 10^{-6}\text{s} \cdot D^{2.006(5)}$ and $\tau^{LDm}_{n_{eig}=10}(D) = 5.1(2) \cdot 10^{-6}\text{s} \cdot D^{2.006(2)}$. One-dimensional BEC potential with anharmonicity $k_n = 1$ has been diagonalized for the area $|x| \leq 10$, $\Delta = \frac{20}{D}$, rotation parameter $r = 0$, propagation time $\epsilon = 0.1$ and the level of effective action $p = 21$.



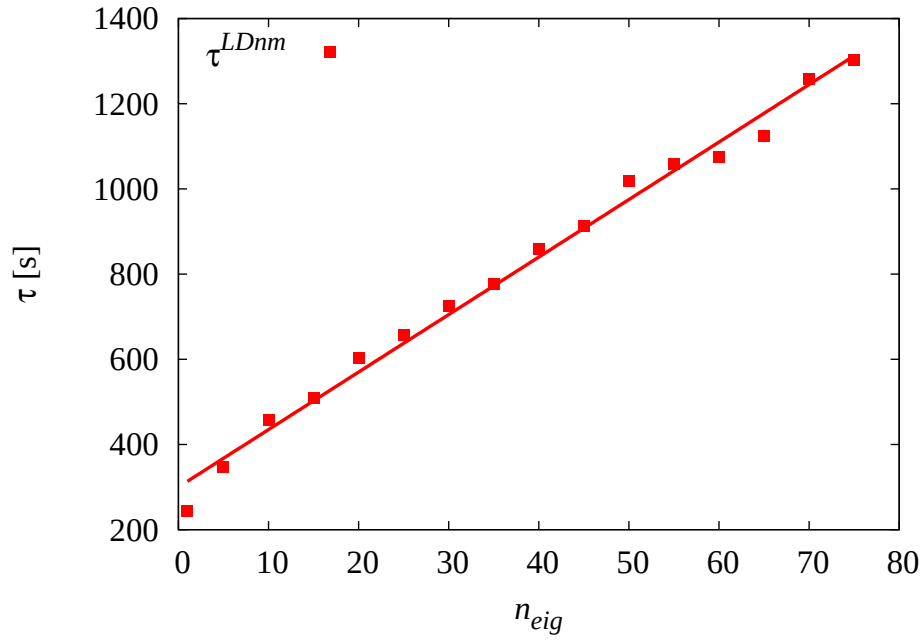
3.8: The runtime of Lanczos diagonalization ($LDnm$) algorithm without the storage of the matrix while calculating 1 and 10 eigenvalues as a function of the matrix size, fitted by power-laws $\tau^{LDnm}_{n_{eig}=1}(D) = 1.4(1) \cdot 10^{-4}\text{s} \cdot D^{1.93(1)}$ and $\tau^{LDnm}_{n_{eig}=10}(D) = 1.34(3) \cdot 10^{-4}\text{s} \cdot D^{2.004(3)}$. Other system parameters are same as at the previous figure.



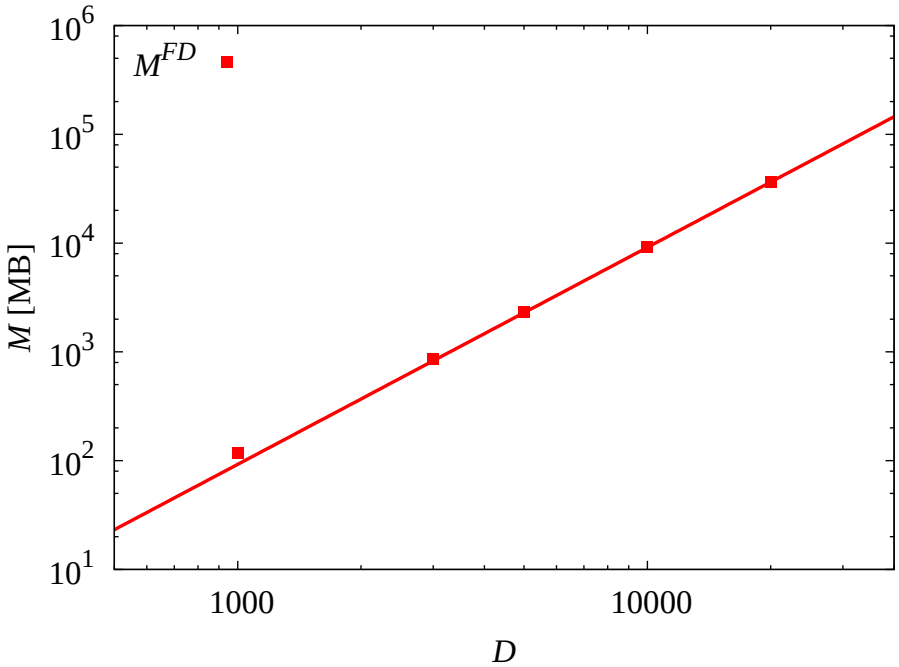
3.9: The runtime of Lanczos diagonalization ($LDnm$) algorithm without the storage of the matrix, while calculating one eigenvalue for functions with different effective action level p, as a function of the matrix size fitted by power-laws $\tau_{p=2}^{LDnm}(D) = 2.21(6)\cdot 10^{-6}\text{s} \cdot D^{2.000(3)}$, $\tau_{p=4}^{LDnm}(D) = 2.68(4)\cdot 10^{-6}\text{s} \cdot D^{2.000(2)}$ and $\tau_{p=6}^{LDnm}(D) = 3.88(3)\cdot 10^{-6}\text{s} \cdot D^{1.993(1)}$. One-dimensional BEC potential with anharmonicity $k_n = 1$ has been diagonalized for the area $|x| \leq 10$, $\Delta = \frac{20}{D}$, rotation parameter $r = 0$ and propagation time $\epsilon = 0.1$.



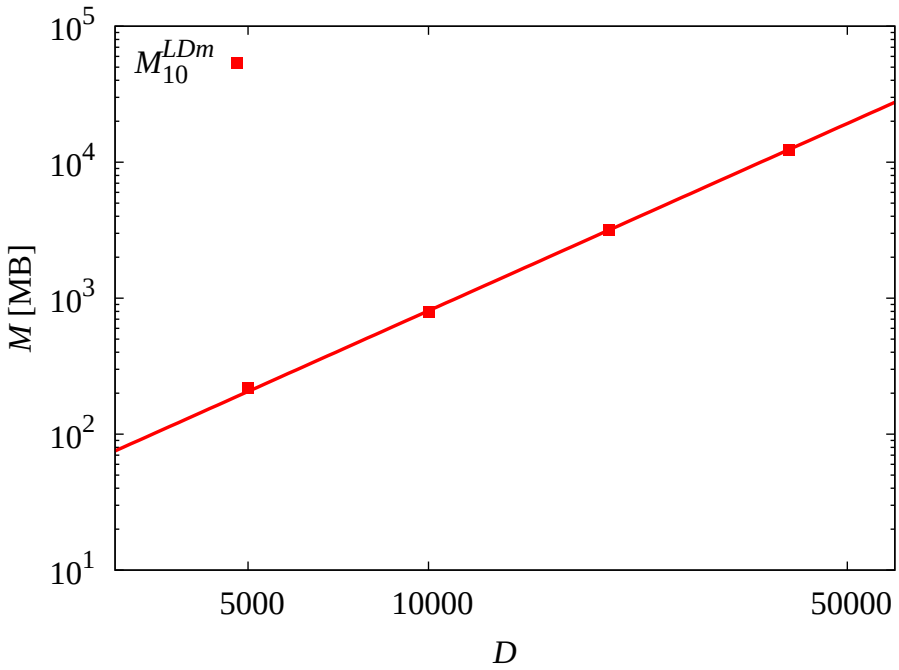
3.10: The runtime of Lanczos diagonalization (LDm) algorithm with the storage of the matrix as a function of the desired number of eigenvalues fitted linearly $\tau^{LDm}(n_{eig}) = 0.103(4)\text{s} \cdot n_{eig} + 15.7(2)\text{s}$. One-dimensional BEC potential has been diagonalized with the matrix size $D = 2000$ for the area $|x| \leq 10$, $\Delta = 0.01$, anharmonicity $k_n = 10^{-3}$, rotation parameter $r = 0$, propagation time $\epsilon = 0.1$ and the level of effective action $p = 21$.



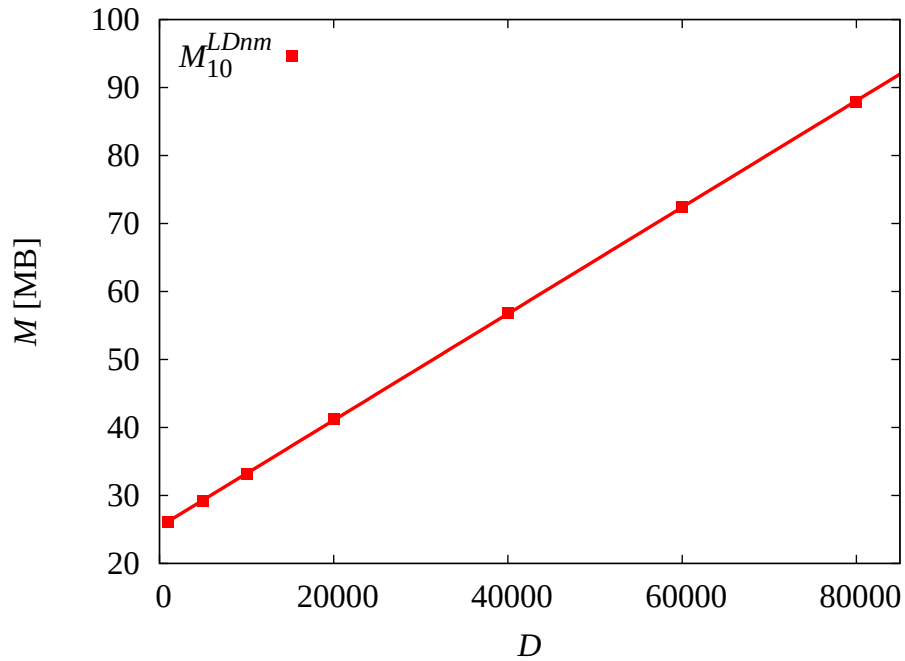
3.11: The runtime of Lanczos diagonalization ($LDnm$) algorithm without the storage of the matrix as a function of the desired number of eigenvalues fitted linearly $\tau^{LDnm}(n_{eig}) = 13.5(4)\text{s} \cdot n_{eig} + 300(20)\text{s}$. One-dimensional BEC potential has been diagonalized with the matrix size $D = 2000$ for the area $|x| \leq 10$, $\Delta = 0.01$, anharmonicity $k_n = 10^{-3}$, rotation parameter $r = 0$, propagation time $\epsilon = 0.1$ and the level of effective action $p = 21$.



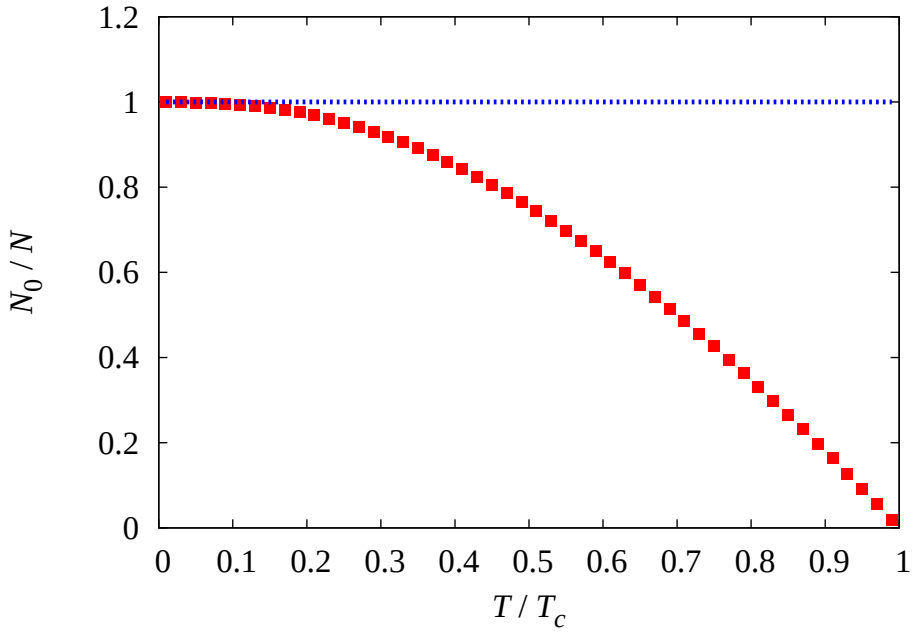
3.12: The memory usage of a full diagonalization (FD) program for solving the eigenproblem of the evolution matrix A, for one-dimensional system, as a function of the linear dimension D of the matrix being diagonalized, fitted to a power-law $M^{FD}(D) = 9.6(3) \cdot 10^{-5}\text{MB} \cdot D^{1.995(3)}$. One-dimensional BEC potential has been diagonalized for the area $|x| \leq 10$, $\Delta = \frac{20}{D}$, anharmonicity $k_n = 0$, rotation parameter $r = 0$, propagation time $\epsilon = 0.1$ and the level of effective action $p = 21$.



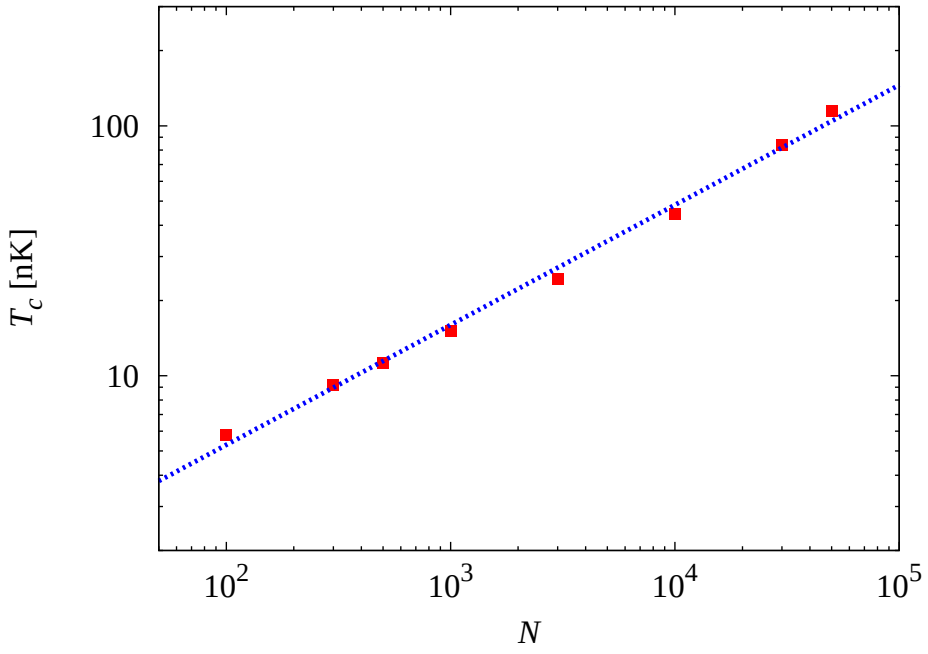
3.13: The memory usage of a Lanczos algorithm with the storage of the matrix (LDm) program for solving the eigenproblem of the evolution matrix A, for one-dimensional system, as a function of the linear dimension D of the matrix being diagonalized, fitted to a power-law $M^{LDm}(D) = 1.05(6) \cdot 10^{-5}\text{MB} \cdot D^{1.971(6)}$. One-dimensional BEC potential has been diagonalized with $\Delta = 0.1$ for the area $|x| \leq D\Delta/2$, anharmonicity $k_n = 0$, rotation parameter $r = 0$, propagation time $\epsilon = 0.1$ and the level of effective action $p = 21$.



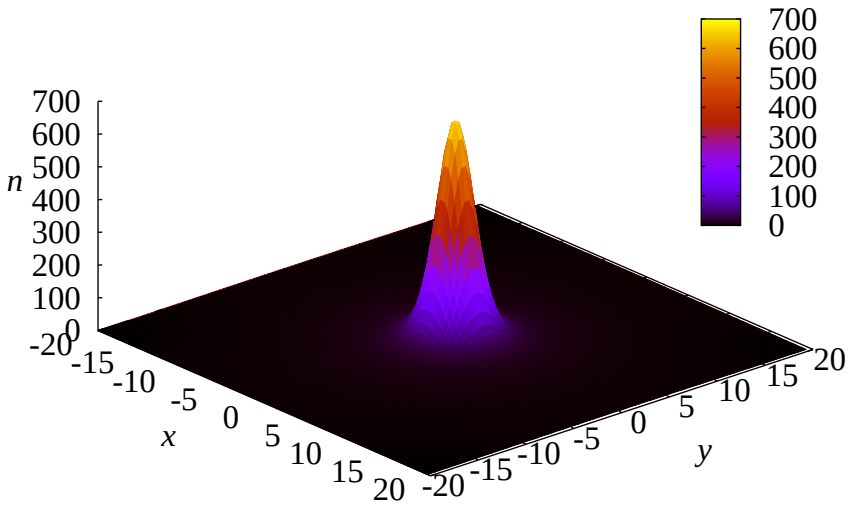
3.14: The memory usage of a Lanczos algorithm without the storage of the matrix ($LDnm$) program for solving the eigenproblem of the evolution matrix A, for one-dimensional system, as a function of the linear dimension D of the matrix being diagonalized, fitted linearly $M^{LDnm}(D) = 7.83(3) \cdot 10^{-4}\text{MB} \cdot D + 25.4(9)\text{MB}$. Other system parameters are same as at the previous figure.



3.15: The fraction of $N = 10000$ particles which occupy the ground state as a function of the temperature, lower than condensation temperature, $T_c = 44.567$ nK. Two-dimensional BEC potential has been diagonalized. Simulation used matrix of dimension $D = 10000$ for the area $|x|, |y| \leq 20$, $\Delta = 0.4$, propagation time $\epsilon = 0.1$, anharmonicity $k_n = 0.00195$, rotation parameter $r = 0.95833$ and the level of effective action $p = 21$.



3.16: The condensation temperature T_c dependence on the number of bosons N in the system fitted to a power-law $T_c(N) = 0.58(6)\text{nK} \cdot N^{0.48(2)}$. Two-dimensional BEC potential has been diagonalized. Simulation used matrix of dimension $D = 10000$ for the area $|x|, |y| \leq 20$, $\Delta = 0.4$, propagation time $\epsilon = 0.1$, anharmonicity $k_n = 0.00195$, rotation parameter $r = 0.95833$ and the level of effective action $p = 21$.



3.17: The particle density n dependence on the position in $x-y$ plane for a gas of $N = 10000$ ideal bosons in rotating anharmonic potential at the temperature $T = 0.01T_c$, $T_c = 44.567$nK. Two-dimensional BEC potential has been diagonalized. Simulation used matrix of dimension $D = 10000$ for the area $|x|, |y| \leq 20$, $\Delta = 0.4$, propagation time $\epsilon = 0.1$, anharmonicity $k_n = 0.00195$, rotation parameter $r = 0.95833$ and the level of effective action $p = 21$.

4

Discussion

Numerical results presented in the previous chapter illustrate and verify the theoretically derived complexities of studied algorithms, as well as the previously derived errors of the applied method.

Figure 3.1 shows that effective action method for $\epsilon < 1$ and different values of p parameter gives high precision, proportional to ϵ^p. Most of the simulations used in this thesis have the values $p = 21$ and $\epsilon = 0.1$. The plot shows that for these values the method introduces smaller error than the usual, which originates from the computer representation of real (double) numbers, therefore for $p = 21$ the error saturates around $\Delta E_0 \sim 10^{-15}$. This implies that the chosen method for the evolution matrix elements calculation does not introduce a significant error and matrix elements can be taken as exact.

Figure 3.2 shows the calculated values of energy levels with full diagonalization algorithm. Since the discretization and double precision are introduced in the calculation, errors in energy values are obtained. Figure 3.3 shows energy levels calculated with Lanczos algorithm. Comparing the error of these values with the errors on the previous plot, one finds the two very similar. Thus, for the purpose of this thesis, the correctness of the Lanczos algorithm is considered to be equal to that of the full diagonalization algorithm. In practice, this should always be checked. A good criterion is the comparison of numerically obtained density of states with its semiclassical approximation [8].

Figures 3.4 and 3.5 show that state functions calculations (as evolution matrix eigenvectors) both with full diagonalization and Lanczos, algorithms give results which excellently match the analytic results, when they are known[19]. This implies that Lanczos algorithm successfully calculates particle eigenstates as well.

As it is shown on Figure 3.6, full diagonalization algorithm time complexity is $\tau^{FD} = O(D^3)$, while Lanczos diagonalization has much more favorable complexity $\tau^{LD} = O(n_{eig}D^2)$ (Figures 3.7–3.11). In case when all the eigenvalues are required, both algorithms are of the same time complexity class. For applications in physics, it is usually enough to calculate only a few of the first eigenvalues and therefore Lanczos algorithm can significantly speed up the problem solving. If the number of desired eigenvalues is constant (e.g. $n_{eig} = 10$) the relation $\tau^{LD} \sim (\tau^{FD})^{2/3}$ holds, and the time complexities for different number of dimensions are given in Table 4.1.

[19]Notice that state functions are calculated for the levels with precisely calculated energies.

time complexity τ	$d=1$	$d=2$	$d=3$
full diagonalization algorithm (FD)	$O(L^3)$	$O(L^6)$	$O(L^9)$
Lanczos algorithm (LD)	$O(L^2)$	$O(L^4)$	$O(L^6)$

Table 4.1: The time complexity for full diagonalization and Lanczos algorithm with constant number of desired eigenvalues in one, two and three dimensions in which coordinates take discrete values $\{-\frac{L}{2}\Delta, -(\frac{L}{2}-1)\Delta, ..., (\frac{L}{2}-2)\Delta, (\frac{L}{2}-1)\Delta\}$ and Δ represents discretized space step.

Memory complexity for full diagonalizaion algorithm is $M^{FD} = O(D^2)$, the same as for Lanczos algorithm with the storage of the matrix being diagonalized $M^{LDm} = O(D^2)$, while Lanczos algorithm without the storage of the matrix has more favorable memory complexity, $M^{LDnm} = O(n_{eig}D)$. In the case all the eigenvalues are required, this complexity becomes the previously mentioned $O(D^2)$. Lanczos algorithm without the storage of the matrix is, even though of same time complexity class, in principle slower than its analogue with the storage of the matrix, because the matrix elements have to be calculated in each iteration separately. Still, in the case when the memory is a limiting factor, Lanczos algorithm without the storage of the matrix can come out as the only solution to the problem. If the number of desired eigenvalues is kept constant (e.g. $n_{eig} = 10$) the relation $M^{LDnm} \sim (M^{FD})^{1/2} \sim (M^{LDm})^{1/2}$ holds, which is proven in Figures 3.12–3.14. The memory complexities for different number of dimensions are given in Table 4.2.

memory complexity M	$d=1$	$d=2$	$d=3$
full diagonalization algorithm (FD)	$O(L^2)$	$O(L^4)$	$O(L^6)$
Lanczos algorithm with the storage of the matrix (LDm)	$O(L^2)$	$O(L^4)$	$O(L^6)$
Lanczos algorithm without the storage of the matrix (LDnm)	$O(L)$	$O(L^2)$	$O(L^3)$

Table 4.2: The memory complexity for full diagonalization and Lanczos algorithms with constant number of desired eigenvalues in one, two and three dimensions in which coordinates take discrete values $\{-\frac{L}{2}\Delta, -(\frac{L}{2}-1)\Delta, ..., (\frac{L}{2}-2)\Delta, (\frac{L}{2}-1)\Delta\}$ and Δ represents discretized space step.

The studies of algorithm time and memory complexity and their practical implementation is the key element for the application of numerical simulations. Based on the results presented in this chapter, it is possible to choose the optimal algorithm for BEC investigation (as well as for other phenomena which require exact diagonalization) depending on the system parameters and available computer resources. Besides the principal power-laws shown, the crucial elements in optimization are the actual values of constant prefactors, which can happen to favor a certain algorithm for a given set of parameters, despite the opposite expectations of the general analysis (e.g. $O(L^2)$ versus $O(L^3)$).

The success of the application of the developed Lanczos diagonalization algorithm onto the studied system of a gas of ideal ^{87}Rb bosons is shown on Figures 3.14–3.16, where the obtained dependencies agree with those known from the literature: ground state occupancy during the system cooling below the condensation temperature, rise of the condensation temperature with the number of bosons in the system, particle density sharp peak in the center of the system for temperature below the condensation temperature.

5

Conclusions

This thesis studies numerical simulations of Bose-Einstein condensates in rotating magneto-optical trap, as well as their crucial numerical step in calculating the condensation temperature in the diagonalization of space-discretized evolution operator. Accuracy, time and memory complexity are compared for full diagonalization algorithm and Lanczos algorithm in two versions, with (LDm) and without ($LDnm$) the storage of the matrix being diagonalized. Full diagonalization program used LAPACK library functions (the code is given in Appendix A.1), while for Lanczos diagonalization a C/C++ program has been developed with the two described options (the code is given in Appendix A.2). The code of effective action function for the level $p = 6$ is presented in Apendix A.3. The developed Lanczos algorithm program has been applied to the gas of ideal bosons of ^{87}Rb and the obtained dependencies correspond to the theoretical predictions and known experimental results.

For the investigated problem (diagonalization of harmonic potential with quartic anharmonicity), the correctness of Lanczos diagonalization is found to be similar to full diagonalization and, in both cases, dependent on discretization parameters and computer numerical precision. Lanczos algorithm is the most favorable in the applications requiring a small number of eigenvalues. Then this algorithm has a lower class of time complexity compared to full diagonalization algorithm which is illustrated with the relation

$$\tau^{LD} \sim (\tau^{FD})^{2/3}.$$

In case when memory is the limiting factor, Lanczos algorithm without the storage of the matrix being diagonalized has a great advantage since its memory complexity is lower than other algorithms

$$M^{LDnm} \sim (M^{FD})^{1/2}.$$

Analytical and numerical results presented in this thesis can be directly used for the optimization of the algorithm choice for solving of different problems that require exact diagonalization of large matrices. In addition to this, there are many interesting questions for further investigations, such as the analysis of the dependence of the energy level calculation error on discretization parameter values and the convergence of the problem, especially for Lanczos diagonalization algorithm. After simple simulation code changes, Lanczos algorithm can easily be applied to other problems requiring efficient diagonalization.

Appendix A

Program Code

A.1 C/C++ Implementation of Full Diagonalization Algorithm

Implementation of full diagonalization algorithm for solving the eigenproblem of the evolution operator matrix A in $d = 1$ uses LAPACK [17] package functions. To run the program, the following values are required: the discretization parameters L and $L\Delta$, the evolution time ϵ, BEC potential parameters $A = 1 - r^2$ and anharmonicity k_n. Effective action function `S_eff()` has been left out and should be additionally defined within the program. An example of such a function is given in Appendix A.3.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include <float.h>
#include <time.h>

int main(int argc, char **argv)
{

   double S_eff(double, double, double, double, double);
   int *ivector(long, long);
   double *dvector(long, long);
   void free_ivector(int *, long, long);
   void free_dvector(double *, long, long);

   int *IWORK, INFO;
   long NF, LWORK, LIWORK, L, i, k, maxenlev;
   double delta, xi, xk, *M, *d, *v, A, kn, eps, Pi, temp, r, g;
   FILE *psi;
   char JOBZ, UPLO;
   time_t t_exec;

   if(argc != 7)
   {
```

```
   fprintf(stderr, "Usage: %s psi L L*delta eps A kn\n\n\n", argv[0]);
   exit(EXIT_FAILURE);
}

t_exec = time(NULL);
psi = fopen(argv[1], "w+");
Pi = 4 * atan(1);
L = atol(argv[2]);
delta = atof(argv[3]) / L;
eps = atof(argv[4]);
A = atof(argv[5]);
kn = atof(argv[6]);
M = dvector(0, 2 * L * 2 * L - 1);
JOBZ = 'V';
UPLO = 'U';
NF = 2 * L;
LIWORK = 3 + 5 * NF;
LWORK = 1 + 6 * NF + 2 * NF * NF;
d = dvector(0, NF - 1);
v = dvector(0, LWORK - 1);
IWORK = ivector(0, LIWORK - 1);
INFO = 0;

for(i = 0; i < 2 * L; i++)
{
   xi = (i - L) * delta;

   for(k = i; k < 2 * L; k++)
   {
      xk = (k - L) * delta;
      M[i + 2 * L * k] = exp(-S_eff(0.5 * (xi + xk), xk - xi, A, kn, eps));
   }
}

t_exec -= time(NULL);
t_exec = time(NULL);
dsyevd_(&JOBZ, &UPLO, &NF, M, &NF, d, v, &LWORK, IWORK, &LIWORK, &INFO);
t_exec -= time(NULL);
free_ivector(IWORK, 0, LIWORK - 1);
free_dvector(v, 0, LWORK - 1);

maxenlev = 0;

for(i = 0; i < NF / 2; i++)
{
   temp = d[i];
   temp = temp < 0 ? DBL_MAX : (maxenlev++, - log(temp * delta
            / sqrt(2 * Pi * eps)) / eps);
   d[i] = d[NF - 1 - i];
   d[i] = d[i] < 0 ? DBL_MAX : (maxenlev++, - log(d[i] * delta
            / sqrt(2 * Pi * eps)) / eps);
```

```
      d[NF - 1 - i] = temp;
   }

   for(i = 0; i < maxenlev; i++)
   {
      fprintf(psi, "%d\n1.16le\n", i, d[i]);
   }


   fprintf(psi, "\n");
   fprintf(psi, "Eigenvectors:\n");
   for(k = 0; k < 2 * L; k++)
   {
        fprintf(psi, "%d\t", k);
   for(i = 0; i < 10/*maxenlev; i++)
   {
        fprintf(psi, "%1.16le\t", M[k + 2 * L * (2 * L - 1 - i)]);
   }

      fprintf(psi, "\n");
   }

   fclose(psi);
   free_dvector(M, 0, 2 * L * 2 * L - 1);

   exit(EXIT_SUCCESS);
}
```

A.2 C/C++ Implementation of Lanczos Diagonalization Algorithm

Implementation of Lanczos diagonalization algorithm for solving the eigenproblem of the evolution operator matrix A in $d = 1$ offers the options of the storage of the matrix in the memory and calculating the matrix elements in each iteration [18]. To run the program the following values are required: the maximal number of iterations (typically double the number of desired eigenvalues), the discretization parameters D and Δ, the number of desired eigenvalues and eigenvectors, evolution time ϵ, BEC potential parameters $A = 1 - r^2$ and anharmonicity k_n, as well as the option with/without matrix storage. Effective action function `S_eff()` has been left out and should be additionally defined within the program. An example of such a function is given in Appendix A.3.

```
#include <stdlib.h>
#include <math.h>
#include <stdio.h>
#include <gsl/gsl_math.h>
#include <gsl/gsl_heapsort.h>
```

```
int Dimension;
//size of Hamiltonian matrix
double** QVector;
//Lanczos vectors, size = maxNbIter * Dimension
int QVsize;
//Lanczos vector size
int Index;
//iteration index
int nbIter, maxNbIter;
//number of iterations and maximal number of Lanczos iterations
double *TDdiagonal, *TDupperdiag;
//Tridiagonal matrix, size = maxNbIteration
double *tempTDdiagonal, *tempTDupperdiag;
//Temporary tridiagonal matrix, size = maxNbIteration
int TDsize;
//Tridiagonal matrix size
int nbEigenvalues;
//number of desired egenvalues
double prevLowest;
//previous lowest eigenvalue
double dx, A, kn, eps;
//phi4 parameters
double **tempEigenvector;
//temporary matrix for Eigenvector evaluation, size = <= maxNbIter * maxNbIter
double **Eigenvectors;
//Eigenvectors, size = Dimension * nbEigenvalues
bool useMatrix;
//true = store Hamiltonian values, false = calculate on the fly Hamiltonian values
double **HamiltonianMatrix;
//Hamiltonian matrix, size = Dimension * Dimension

//function evaluating Hamiltonian value
double HamiltonianValue(int i, int j)
{
        if (useMatrix == true) return HamiltonianMatrix[i][j];
        double konst = 1./sqrt(2*eps*4*atan(1.));
        double S = S_eff((i+j-Dimension)*dx/2,(j-i)*dx,A,kn,eps);
        return -konst*exp(-S);
}


//Vector norm
double VectorNorm(double* Vector, int size)
{
        double sum = 0;
        int i;
        for (i = 0; i < size; i++)
        {
                sum += Vector[i] * Vector[i];
        }
```

```
        return sqrt(sum);
}


//normalizes vector
void NormalizeVector(double* Vector, int size)
{
        int i;
        double norm = VectorNorm(Vector, size);
        if (norm == 0) return;
        for (i=0; i<size; i++)
        {
                Vector[i] /= norm;
        }
        return;
}


//realocates for another Lanczos vector
void AddQVector(int nbAdditional)
{
        QVsize += nbAdditional;
        int i;
        for(i = nbAdditional; i > 0; i--)
                QVector[QVsize - i] = (double*) calloc(Dimension, sizeof(double));*/
        return;
}


//Lanczos initialization
void InitializeLanczos()
{
        if (useMatrix)
        {
                double konst = 1./sqrt(2*eps*4*atan(1.));
                double S;
                HamiltonianMatrix = new double* [Dimension];
                for (int i = 0; i < Dimension; i++)
                {
                        HamiltonianMatrix[i] = new double [Dimension];
                        for (int j = 0; j < Dimension; j++)
                        {
                                S = phi4d1p21((i+j-Dimension)*dx/2,(j-i)*dx,A,kn,eps);
                                HamiltonianMatrix[i][j] = -konst*exp(-S);
                        }
                }
        }

        QVector = (double**) malloc((maxNbIter + 2) * sizeof(double *));
        int i;
        QVsize = 3;
```

```
        for(i = 0; i < maxNbIter + 2; i++)
        QVector[i] = (double*) malloc(Dimension * sizeof(double));

        for (i = 0; i < Dimension; i++)
            {
                      QVector[0][i] = rand()-0.5;
        }
        NormalizeVector(QVector[0], Dimension);

        TDdiagonal = (double*) calloc(maxNbIter, sizeof(double));
        TDupperdiag = (double*) calloc(maxNbIter - 1, sizeof(double));
        tempTDdiagonal = (double*) calloc(maxNbIter, sizeof(double));
        tempTDupperdiag = (double*) calloc(maxNbIter - 1, sizeof(double));

        Index = 0;
        TDsize = 0;
        return;
}


//Printing vector
void printVector(double* Vector, int size)
{
        int i;
        for (i = 0; i < size; i++)
        {
                printf("%1.14le\n", Vector[i]);
        }
        printf("\n");

        return;
}


//retun maximal integer out of two given ones
int max(int a, int b)
{
        if (a > b) return a;
        return b;
}


//resizes Tridiagonal matrix;
void TDresize(int size)
{
        TDsize = size;

        return;
}
```

```
//p = A*q                *time consuming!
void HamiltonianTimesVector(double* q, double* p, int size)
{
        int i,j;
        for (i=0; i<size; i++)
        {
                p[i] = 0;
                for (j=0; j<size; j++)
                {
                        p[i] += HamiltonianValue(i,j) * q[j];
                }
        }
        return;
}


//Scalar product (p, q)
double ScalarProduct(double* p, double*q, int size)
{
        int i;
        double sum = 0;
        for (i=0; i<size; i++)
        {
                sum += p[i] * q[i];
        }
        return sum;
}


//q += a*p
void AddLinearCombination(double* q, double a, double* p, int size)
{
        int i;
        for (i=0; i<size; i++)
        {
                q[i] += a * p[i];
        }
}


//Lanczos iteration
void Lanczos(int nbIter)
{
        int size;

        if (Index == 0)
        {
                size = TDsize + max(nbIter, 2);
                TDresize(size);

                HamiltonianTimesVector(QVector[0], QVector[1], Dimension);
```

```
        TDdiagonal[Index] = ScalarProduct(QVector[0], QVector[1], Dimension);
        AddLinearCombination(QVector[1], -TDdiagonal[Index], QVector[0], Dimension);
        NormalizeVector(QVector[1], Dimension);
        HamiltonianTimesVector(QVector[1], QVector[2], Dimension);
        TDupperdiag[Index] = ScalarProduct(QVector[0], QVector[2], Dimension);
        TDdiagonal[Index + 1] = ScalarProduct(QVector[1], QVector[2], Dimension);
}
  else
{
        size = TDsize + nbIter;
        TDresize(size);
}

int i,j;

for (i = Index + 2; i < size; i++)
    {
AddLinearCombination(QVector[i], -TDdiagonal[Index + 1], QVector[i - 1], Dimension);
AddLinearCombination(QVector[i], -TDupperdiag[Index], QVector[i - 2], Dimension);

if (i > 2)
{
        double pom;
        for(j = 0; j < i - 2; j++)
        {
        pom = -ScalarProduct(QVector[i], QVector[j], Dimension);
        AddLinearCombination(QVector[i], pom, QVector[j], Dimension);
        }
}

double norm = VectorNorm(QVector[i], Dimension);
int errNb = 0;
     while (norm < 1e-10)
{
        printf("i  %d  norm %1.18le \n", i, norm);
        errNb++;
        if (errNb > 20)
        {
                printf("Lanczos algorithm cannot converge\n");
                exit(0);
        }

        double tmp = 0;

        for (j = 0; j < Dimension; j++)
        {
                QVector[i][j] = rand()-0.5;//gsl_rng_uniform (random) - 0.5;
        }

        NormalizeVector(QVector[i], Dimension);
```

```
                double* tmpVector;
                tmpVector = (double*) malloc(Dimension * sizeof(double));
                HamiltonianTimesVector(tmpVector, QVector[i], Dimension);
                QVector[i] = tmpVector;

                double pom;
                for(j = 0; j < i; j++)
                {
                        pom = -ScalarProduct(QVector[i], QVector[j], Dimension);
                        AddLinearCombination(QVector[i], pom, QVector[j], Dimension);
                }
                norm = VectorNorm(QVector[i], Dimension);
                }

                NormalizeVector(QVector[i], Dimension);
                Index++;
                AddQVector(1);

                HamiltonianTimesVector(QVector[i], QVector[i + 1], Dimension);

        TDupperdiag[Index] = ScalarProduct(QVector[i - 1], QVector[i + 1], Dimension);
        TDdiagonal[Index + 1] = ScalarProduct(QVector[i], QVector[i + 1], Dimension);
        }

}


//compare function for heap sort
int compare_doubles (const void * a, const void * b)
{
        double aa = *((double*)a);
        double bb = *((double*)b);
        if (aa > bb)
                return 1;
        else if (aa < bb)
                return -1;
        else
                return 0;
}


//sorts eigenvalues in diagonal matrix from the lowest to the highest
void sortDiagonal()
{
        gsl_heapsort (tempTDdiagonal, TDsize, sizeof(double), compare_doubles);

        return;
}


//Diagonalizes Tridiagonal matrix
```

```
void Diagonalize(bool vectors)
{
        int counter;

        for (counter = 0; counter < TDsize; counter++)
        {
                tempTDdiagonal[counter] = TDdiagonal[counter];
                if (counter != Dimension - 1)
                tempTDupperdiag[counter] = TDupperdiag[counter];
        }


        int ReducedDimension = TDsize - 1;
        double Cos;
        double Sin;
        double Theta;
        double T, R, P, F, B;
        int maxIter = 1000;

          for (int i = 0; i < ReducedDimension; i++)
            {
                int iter = 0;
                while (iter < maxIter)
                {
                  // find block matrices so that QL algorithm will be applied
                // on submatrix from i to j
                  int j = i;
                  bool Flag = false;
                  while ((j < ReducedDimension) && (Flag == false))
                    {
                      double d2 = fabs(tempTDdiagonal[j]) + fabs(tempTDdiagonal[j + 1]);
                      if ((d2 + fabs(tempTDupperdiag[j])) == d2)
                Flag = true;
                      else
                j++;
                    }
                  // if i != j, i-th eigenvalue has not been obtained yet,
                // apply diagonalization on submatrix
                  if (j != i)
                    {
                      iter++;
                      // evaluate shift
                      Theta = (tempTDdiagonal[i + 1] - tempTDdiagonal[i])
                        / (2.0 * tempTDupperdiag[i]);
                      R = sqrt (1.0 + Theta * Theta);
                      T = tempTDdiagonal[j] - tempTDdiagonal[i];
                      if (Theta >= 0)
                T += tempTDupperdiag[i] / (Theta + R);
                      else
                T += tempTDupperdiag[i] / (Theta - R);
                      Cos = 1.0;
```

```
                Sin = 1.0;
                P = 0.0;
                // apply shift and conjugation with Jacobi and Givens rotations
                for (int k = j - 1; k >= i; k--)
            {
            F = Sin * tempTDupperdiag[k];
            B = Cos * tempTDupperdiag[k];
            R = sqrt (F * F + T * T);
            tempTDupperdiag[k + 1] = R;
            if (R == 0.0)
            {
            tempTDdiagonal[k + 1] -= P;
            tempTDupperdiag[j] = 0.0;
            k = i - 1;
            }
            else
            {
            Sin = 1.0 / R;
            Cos = Sin * T;
            Sin *= F;
            T = tempTDdiagonal[k + 1] - P;
            R = (tempTDdiagonal[k] - T) * Sin + 2.0 * Cos * B;
            P = Sin * R;
            tempTDdiagonal[k + 1] = T + P;
            T = Cos * R - B;
            // apply transformation to vectors
            if (vectors)
            {
            double tmp;
            for (int n = 0; n < TDsize; n++)
            {
            tmp = tempEigenvector[n][k + 1];
            tempEigenvector[n][k + 1] *= Cos;
            tempEigenvector[n][k + 1] += Sin * tempEigenvector[n][k];
            tempEigenvector[n][k] *= Cos;
            tempEigenvector[n][k] -= Sin * tmp;
            }
            }
            }
            }
                  tempTDdiagonal[i] -= P;
                  tempTDupperdiag[i] = T;
                  tempTDupperdiag[j] = 0.0;
                }
              else
                iter = maxIter;
            }
        }

    if (!vectors)
            sortDiagonal();
```

```
        return;
}


//evaluates and prints eigenvecors
void printEigenstates()
{
        Eigenvectors = new double* [nbEigenvalues];

        tempEigenvector = new double* [TDsize];

        for (int i = 0; i < TDsize; i++)
        {
                tempEigenvector[i] = new double[TDsize];
                for (int j = 0; j < TDsize; j++)
                        tempEigenvector[i][j] = 0;
                tempEigenvector[i][i] = 1.0;
        }

        Diagonalize(true);

        int ReducedDim = TDsize - 2;
          double tmp;
          int MinPos;
          double MinValue;
          for (int i = 0; i <= ReducedDim; i++)
            {
                      MinPos = TDsize - 1;
                      MinValue = tempTDdiagonal[MinPos];
                      for (int j = ReducedDim; j >= i; j--)
                if (tempTDdiagonal[j] < MinValue)
                {
                          MinValue = tempTDdiagonal[j];
                          MinPos = j;
                }
                      tmp = tempTDdiagonal[i];
                      tempTDdiagonal[i] = MinValue;
                      tempTDdiagonal[MinPos] = tmp;
                for (int ii = 0; ii < TDsize; ii++)
                {
                              tmp = tempEigenvector[ii][i];
                              tempEigenvector[ii][i] = tempEigenvector[ii][MinPos];
                              tempEigenvector[ii][MinPos] = tmp;
                }
            }

          double* TmpCoefficents;
        TmpCoefficents = new double [TDsize];
          for (int i = 0; i < nbEigenvalues; ++i)
            {
```

```
                    for (int j = 0; j < TDsize; ++j)
                      TmpCoefficents[j] = tempEigenvector[j][i];
                  Eigenvectors[i] = new double [Dimension];
              int ii;
                    for (ii = 0; ii < Dimension; ii++)
              Eigenvectors[i][ii] = QVector[0][ii] * tempEigenvector[0][i];
              for (ii = 0; ii < TDsize - 1; ii++)
              AddLinearCombination(Eigenvectors[i], TmpCoefficents[ii+1],
                                   QVector[ii+1], Dimension);
          }


        for (int ii = 0; ii < nbEigenvalues; ii++)
        {
                NormalizeVector(Eigenvectors[ii], Dimension);
        }


        for (int j = 0; j < Dimension; j++)
        {
        printf("%d\t", j);
         for (int i = 0; i < nbEigenvalues; i++)
        {
                printf("%1.18le\t", Eigenvectors[i][j]);
        }
        printf("\n");
        }

}



int main(int argc, char** argv)
{
        int i,j;

        if (argc!=9)
        {
                printf("usage: max number of iterations, matrix dimension,
                        number of eigenvalues, dx, eps, A, kn, useMatrix 1-yes 0-no\n");
                return 0;
        }
        else
        {
                maxNbIter = (int) atol(argv[1]);
                Dimension = (int) atol(argv[2]);
                nbEigenvalues = (int) atol(argv[3]);
                dx = atof(argv[4]);
                eps = atof(argv[5]);
                A = atof(argv[6]);
                kn = atof(argv[7]);
```

```
        if (argv[8][0] == '0') useMatrix = false; else useMatrix = true;
    }

    InitializeLanczos();
    Lanczos(nbEigenvalues + 2);
    double Precision = 1.0;
    int currNbIter = nbEigenvalues + 2;
    prevLowest = 1e30;
     while ((Precision > 1e-14) && (currNbIter++ < maxNbIter))
    {
        Lanczos(1);
        Diagonalize(false);
          Precision = fabs((prevLowest - tempTDdiagonal[nbEigenvalues-1])
                       / prevLowest);
          prevLowest = tempTDdiagonal[nbEigenvalues-1];
    }
     if (currNbIter >= maxNbIter)
    {
          printf("too many Lanczos iterations\n");
          exit(0);
    }
    printf("Eigenvalues:\n");
    double eig;
      for (i = 0; i < nbEigenvalues; ++i)
        {
          eig = -log(-dx*(tempTDdiagonal[i]))/eps;
          printf("%d\t%1.14le\n", i, eig);
        }

    printf("\n");

    printf("Eigenvectors:\n");
    printEigenstates();

    for (i=QVsize-1; i>=0; i--)
        free(QVector[i]);
    free(QVector);
    free(TDdiagonal);
    free(TDupperdiag);
    free(tempTDdiagonal);
    free(tempTDupperdiag);

    return 0;

}
```

A.3 C/C++ Implementation of the Effective Action Function of Order $p = 6$

An example of effective action function for one-dimensional anharmonic oscillator $V = \frac{1}{2}Ax^2 + \frac{k_n}{24}x^4$ of order $p = 6$ is given in this section. Function parameters are: coordinate value, discretization step Δ, $A = 1 - r^2$, anharmonicity k_n and evolution time ϵ.

```
double S_eff_1d_p6(double x, double dx, double A, double kn, double eps){

dx2 = dx*dx;
dx4 = dx2*dx2;
dx6 = dx2*dx4;
dx8 = dx4*dx4;
x2 = x*x;
x4 = x2*x2;
x6 = x3*x3;
x7 = x4*x3;
x8 = x4*x4;
eps2 = eps*eps;
eps3 = eps*eps2;
eps4 = eps2*eps2;
eps5 = eps3*eps2;
kn2 = kn*kn;
kn3 = kn*kn2;
A2 = A*A;

return (0.5*dx2)/eps + eps*(0.00052083333333333333*dx4*kn +
        0.0020833333333333333*dx2*eps*kn - 2.4112654320987654e-7*dx6*eps2*kn2 -
        2.8935185185185185e-6*dx4*eps3*kn2 + 0.5*A*x2 + 0.041666666666666667*kn*x4 +
        eps*(0.083333333333333333*A + 0.041666666666666667*kn*x2) +
        0.041666666666666667*dx2*(A + 0.5*kn*x2) +
        0.25*dx2*eps3*(-0.00079365079365079365*A*kn - 0.0013888888888888889*kn2*x2) +
        0.0625*dx4*eps2*(-0.00039682539682539683*A*kn - 0.00069444444444444444*kn2*x2)
        + 0.25*dx2*eps2*(-0.0027777777777777778*A2 - 0.011111111111111111*A*kn*x2 -
        0.0020833333333333333*kn2*x4) +  eps3*(-0.0027777777777777778*A2 -
        0.011111111111111111*A*kn*x2 - 0.0020833333333333333*kn2*x4) +
        eps2*(0.0041666666666666667*kn - 0.041666666666666667*A2*x2 -
        0.013888888888888889*A*kn*x4 - 0.0011574074074074074*kn2*x6) + 0.25*dx2*eps4*
        (0.00006613756613756613756*A3 - 0.00006448412698412698412*kn2 +
        0.0015873015873015873*A2*kn*x2 + 0.0009246031746031746*A*kn2*x4 +
        0.0001157407407407407407*kn3*x6) + eps5*(0.0001763668430335097*A3 -
        0.00003885582010582010582*kn2 + 0.0021494708994708994708*A2*kn*x2 +
        0.0012400793650793650794*A*kn2*x4 + 0.00015432098765432098765*kn3*x6) +
        eps4*(-0.00059523809523809524*A*kn + 0.0041666666666666667*A3*x2 -
        0.00086805555555555556*kn2*x2 + 0.0034722222222222222222*A2*kn*x4 +
        0.00081018518518518518519*A*kn2*x6 + 0.00005787037037037037037*kn3*x8));
}
```

References

[1] L. Pitaevski, S. Stringari, *Bose–Einstein Condensation*, Oxford University Press (2003).

[2] C. J. Pethick, H. Smith, *Bose–Einstein Condensation in Dilute Gases*, Cambridge University Press (2002).

[3] I. Bloch, J. Dalibard, W. Zwerger, Rev. Mod. Phys. **80**, 885 (2008).

[4] K. B. Davis, M.-O. Mewes, M. R. Andrews, N. J. van Druten, D. S. Durfee, D. M. Kurn, W. Ketterle, Phys. Rev. Lett. **75**, 3969 (1995).

[5] M. H. Anderson, J. R. Ensher, M. H. Matthews, C. E. Wieman, and E. A. Cornell, Science **269**, 198 (1995).

[6] S. Kling, A. Pelster, Phys. Rev. A **76**, 023609 (2007).

[7] I. Vidanović, A. Bogojević A. Balaž, Phys. Rev. E **80**, 066705 (2009).

[8] I. Vidanović, A. Bogojević, A. Balaž, A. Belić, Phys. Rev. E **80**, 066706 (2009).

[9] A. Bogojević, I. Vidanović, A. Balaž, A. Belić, Phys. Lett. A **372**, 3341 (2008).

[10] A. Bogojević, A. Balaž, A. Belić, Phys. Rev. Lett. **94**, 180403 (2005).

[11] A. Bogojević, A. Balaž, A. Belić, Phys. Rev. B **72**, 036128 (2005).

[12] A. Aftalion, *Vortices in Bose–Einstein Condensates*, Birkhäuser, Boston (2006).

[13] A. L. Fetter, Rev. Mod. Phys. **81**, 647 (2009).

[14] V. Bretin, S. Stock, Y. Seurin, J Dalibard, Phys. Rev. Lett. **92**, 050403 (2004).

[15] Y Saad, *Numerical Methods for Large Eigenvalue Problems*, Manchester University Press (1992).

[16] I. Milošević, *Vektorski prostori i elementi vektorske analize*, Univerzitet u Beogradu (1997).

[17] LAPACK numerical library, `http://www.netlib.org/lapack/`

[18] C/C++ Lanczos algorithm implementation `https://svn.scl.rs/lanczos/`