Bachelor Thesis Report
FUFX02-12-01

Configuration-Interaction Methods and Large-scale Matrix Diagonalization

Pontus Hansson
Joakim Löfgren
Karin Skoglund Keiding
Simon Vajedi

May 16, 2012

Abstract

This thesis presents investigations made with matrices representing quantum mechanical many-body systems in order to provide recommendations for a future eigensolver. The matrices are generated using the No-Core Shell Model for bosons (NCSMb) code. Finding the eigenvalues is interesting since these correspond to the energy spectrum of the system of particles. Examples of such systems include nucleons in a nuclei or trapped atomic gases at low temperatures. The eigensolver will be based on the Lanczos algorithm, a method well suited to reduce the eigenvalue-solving of large, sparse matrices.

Investigations are performed concerning the Lanczos method and of how to adapt it in an appropriate way in terms of choice of starting vectors, methods of orthogonalization and of how to accelerate converge. Also, the possibility of Block Lanczos is examined. Guidelines are given in order to make a future eigensolver as effective as possible.

Furthermore, various tests are made investigating the properties of the Hamiltonian matrices such as the dimension of the matrix for different model spaces, the number of nonzero elements and the distribution of the matrix elements among others.

Contents

Acknowledgements

The authors of this report wish to express their sincere gratitude to the supervisors Christian Forssén and Håkan Johansson for their support and guidelines throughout the whole project. Their time investigated for constructive discussions, teaching and reading and correcting this report was invaluable.

Special thanks are also due to Simon Tölle, Christian Forssén, Håkan Johansson and Lucas Platter for providing the programming code NCSMb for calculating matrices for many-boson systems.

1 Introduction

When dealing with systems of many particles, the complexity of modelling the dynamics increases dramatically as the number of particles increases. Solving the Schrödinger equation analytically for such a system is impractical or even impossible, meaning that certain approximations are required.

One approach is to use the configuration interaction method, where the wavefunction representing the many-particle system is expressed as a linear combination of many-particle basis states [1]. These basis states have to respect the appropriate symmetry under particle exchange, and are themselves made up of a linear combination of product states of several single-particle states. The single-particle states are expressed in some appropriate basis, e.g. the harmonic oscillator basis, and form an infinite series expansion that must be truncated to make numerical calculations possible. The Schrödinger equation can thereafter be put on matrix form, with the problem remaining of finding the eigenvalues of the large, sparse Hamiltonian matrix of the system.

Finding the eigenvalues of such a matrix is complicated despite its sparseness, due to the growing dimension of the matrix as the number of particles and the model space increase. However, there are algorithms especially useful for these cases, as they reduce the complexity of the eigenvalue problem. By implementing such an algorithm with appropriate adaptations based on the structure of the matrix, the computation of eigenvalues becomes feasible also for large model spaces.

The retrieved energy eigenvalues correspond to the energy spectrum of the system that is being modeled. Studying the bound-state energy spectrum and the corresponding eigenfunctions is one of the main tools available for scientists to study the physics of many-particle systems. Examples of such systems are many-nucleon states in the nucleus of an atom, many-electron systems in atoms and molecules or trapped atomic gases at low temperatures.

1.1 Specific aims

The ultimate goal of the project has been to develop guidelines and recommendations for a future implementation of an eigensolver, where the matrices are calculated by the code No-Core Shell Model for bosons, NCSMb, currently under development [2]. The eigensolver is based on the Lanczos algorithm, a method particularly well suited to reduce the complexity of finding a few eigenvalues of large, sparse and symmetric matrices.

1.2 Method

Initially, literature studies were conducted on how the matrix elements of the Hamiltonian matrix representing the many-particle problem arise. In order to find statistics of the properties of the matrices, the code NCSMb was run with different values of its input parameters. Lanczos-based algorithms were written in MATLAB, and the particular features of the algorithms were investigated and evaluated.

1.3 Structure of the thesis

In Chapter 2 the reader is given an introduction to quantum-mechanical many-body theory, leading to the matrix formulation of the eigenvalue problem. The numerical approach utilized in No-Core Shell Model is then examined in Chapter 3, where the key features of the NCSMb code are described.

Chapter 4 provides a discussion on the numerical Lanczos methods, with the intention of giving the reader a background to the subsequent Chapter 5, presenting the investigations made concerning the Lanczos method. Thereafter Chapter 6 presents the analysis of the code NCSMb, introduces concepts needed to understand the adaptations made of the code, and describes the adaptations. Chapter 7 presents the investigations made on the properties of the matrices generated with NCSMb.

In Chapter 8 a discussion is held of the results from the investigations, where these are compared and limitations are discussed. Finally, Chapter 9 contains a summary of the results by proposing recommendations for a future implementation of an eigensolver using a Lanczos-based method.

2 Quantum Mechanical Many-Body Systems

Quantum mechanical many-particle systems is an important subject and a lot of research is being done to discover new properties of nuclear and atomic systems. The analytical solutions for such systems do not exist for the most part, and the complexity that arises while increasing the number of interacting particles places high demands on the development of novel approaches and numerical methods. Efficient algorithms are required to attain accurate descriptions of many-particle systems, otherwise the problem may become too computationally intense. In order to obtain any solutions for interacting particles, however, it is crucial to understand the theory behind many-body systems. This chapter will introduce several useful quantum mechanical concepts that are necessary for manipulating many-particle systems, and the intention is to present the elementary terminology and notions for further studies in this field.

In contrast to classical theory, particles of the same species are truly indistinguishable in quantum mechanics, and this will influence the behaviour of interacting particles. An understanding of identical particles is crucial when studying many-body systems, and the quantum mechanical formalism is introduced in Section 2.1 at a basic level. We will find that only two species of particles are possible: fermions and bosons. Furthermore, the many-particle states as well as the general wavefunctions for these systems are formulated, and it is shown how these can be constructed from single-particle states. To delve more into this subject it is encouraged to read the introductory chapters of [3].

It is not unusual in quantum mechanics to express the physical operators in terms of the position operator $\hat{x}$ and the momentum operator $\hat{p}$, and formulate particle states in coordinate representation using the eigenstates of these operators. However, a more convenient description of many-particle systems is obtained in the second quantized form, discussed in Section 2.2. The aim is here to introduce the occupation number representation with creation and annihilation operators to greatly facilitate the treatment of many-body systems. Interested readers who want to learn more about this topic are referred to the works of [4] and [5].

Section 2.3 covers coupled and uncoupled schemes and the difference between them. In short, correlated particles can be treated as a coupled entity by

using an inseparable wavefunction. What scheme is chosen has an impact on how a many-body system is characterized.

2.1 Systems of identical particles

Consider a system of N identical particles. The state of one particle in the system can be described by the single-particle state ket $|\alpha\rangle$, where α denotes a set of single-particle quantum numbers that uniquely identifies the state. The single-particle states are elements of a Hilbert space $\mathscr{H}$. The Hilbert space of the N-particle system is denoted $\mathscr{H}_N$ and is defined as the N-fold tensor product of the single-particle Hilbert space $\mathscr{H}$:

$$\mathscr{H}_N = \mathscr{H} \otimes \mathscr{H} \otimes \ldots \otimes \mathscr{H} = \mathscr{H}^{\otimes N}. \tag{2.1}$$

An element in $\mathscr{H}_N$ is denoted $|\alpha_1\alpha_2\ldots\alpha_N\rangle$ and constitutes a many-particle state. These states are constructed by a linear superposition of product states, which are formed by tensor products of single-particle states and can be written as

$$|\alpha_1\alpha_2\ldots\alpha_N) \equiv |\alpha_1\rangle \otimes |\alpha_2\rangle \otimes \ldots \otimes |\alpha_N\rangle, \tag{2.2}$$

where $\{|\alpha_i\rangle\}$ is an orthonormal basis in $\mathscr{H}$. The completeness of the product states implies that

$$\sum_{\alpha_1\alpha_2\ldots\alpha_N} |\alpha_1\alpha_2\ldots\alpha_N)(\alpha_1\alpha_2\ldots\alpha_N| = 1, \tag{2.3}$$

which is referred to as the *completeness relation*. Different many-particle states are formed by linear combinations of these product states and can thus be written as

$$|\alpha_1\alpha_2\ldots\alpha_N\rangle = \sum_{\sigma\in S_N} c_\sigma |\alpha_{\sigma(1)}\alpha_{\sigma(2)}\ldots\alpha_{\sigma(N)}) \,, \tag{2.4}$$

where S_N is the set of permutations of $\{1,2,\ldots,N\}$, and c_σ is an arbitrary constant for the permutation. Note the difference between the notations $|\ldots\rangle$ and $|\ldots)$; the latter is used for product states, which are linearly combined to form the many-particle states, denoted by an angle bracket. Since single-particle states are used to construct product states, bases of $\mathscr{H}_N$ can ultimately be built using single-particle states as constituents.

There are many combinations of product states that are mathematically valid bases in $\mathscr{H}_N$ but do not correspond to a physical system, an example being the product states $\{|\alpha_1\alpha_2\ldots\alpha_N)\}$. Thus for instance,

$$|\alpha_1\alpha_2\ldots\alpha_N\rangle = |\alpha_1\alpha_2\ldots\alpha_N) \tag{2.5}$$

is not a valid multiparticle state. The reason why many combinations of product states are invalid derives from the indistinguishability of the particles. Since permuting two identical particles does not change the system's physical properties, the probability of either state must therefore be equal. Therefore, if $\psi_{\alpha_1\alpha_2\ldots\alpha_N}(\boldsymbol{x}_1\boldsymbol{x}_2\ldots\boldsymbol{x}_N) = (\boldsymbol{x}_1\boldsymbol{x}_2\ldots\boldsymbol{x}_N|\alpha_1\alpha_2\ldots\alpha_N\rangle$ is the wavefunction of the original many-particle system, where $\boldsymbol{x}_i$ is the set of coordinates of the quantum numbers α_i, then

$$\begin{aligned} & \left|\,\psi_{\alpha_1\alpha_2\ldots\alpha_N}(\boldsymbol{x}_1\boldsymbol{x}_2\ldots\boldsymbol{x}_N)\,\right|^2 = \left|\,\psi_{\alpha_2\alpha_1\ldots\alpha_N}(\boldsymbol{x}_2\boldsymbol{x}_1\ldots\boldsymbol{x}_N)\,\right|^2 \\ \Leftrightarrow \quad & \psi_{\alpha_1\alpha_2\ldots\alpha_N}(\boldsymbol{x}_1\boldsymbol{x}_2\ldots\boldsymbol{x}_N) = e^{i\theta}\psi_{\alpha_2\alpha_1\ldots\alpha_N}(\boldsymbol{x}_2\boldsymbol{x}_1\ldots\boldsymbol{x}_N) \end{aligned} \tag{2.6}$$

Interchanging the same particles again would result in the original wavefunction, hence

$$e^{2i\theta} = 1, \tag{2.7}$$

which holds for $\theta = 0$ and $\theta = \pi$. Thus, when permuting two particles the resulting state is obtained by multiplication of either 1 or -1, and this implies that many-particle states must be either *symmetric* or *antisymmetric*, respectively, under interchange of two particles. Symmetric particles are called *bosons*, and antisymmetric particles are *fermions*, and only these two species of particles are found in nature. To obtain a general expression of their respective many-particle states we first introduce the permutation operator P_{ij}, which permutes the states of particles i and j,

$$\begin{aligned} P_{ij}|\alpha_1\alpha_2\ldots\alpha_i\ldots\alpha_j\ldots\alpha_N) &= |\alpha_1\alpha_2\ldots\alpha_j\ldots\alpha_i\ldots\alpha_N) \\ &= \pm|\alpha_1\alpha_2\ldots\alpha_i\ldots\alpha_j\ldots\alpha_N). \end{aligned} \tag{2.8}$$

Hence the eigenvalues of the permutation operator are $+1$ and -1, and the eigenstates are bosonic and fermionic, respectively. In either case the physical properties of the states do not change when applying the permutation operator. From the properties of the permutation operator it is clear that

$$P_{ij} = P_{ji}\,, \qquad P_{ij}^2 = 1. \tag{2.9}$$

The antisymmetrizer $\mathcal{A}$ and the symmetrizer $\mathcal{S}$ are used in order to obtain the correct symmetry of the N-particle states, and they are defined as

$$\mathcal{A} = \frac{1}{N!}\sum_{ij}(-1)^{\pi(ij)}P_{ij} \quad \text{and} \quad \mathcal{S} = \frac{1}{N!}\sum_{ij}P_{ij}, \tag{2.10}$$

where the sums run over all $N!$ permutations of the N-particle states, and $(-1)^\pi$ is the *parity* (the sign) of the permutation and specifies whether the permutation π is even or odd. The permutation is even if an even number of pairs are interchanged, otherwise it is odd.

In the case of fermions the eigenvalue of the permutation operator P_{ij} is as mentioned -1 and the states are therefore antisymmetric. The effect of this

is that two particles in a system cannot have the same quantum numbers, and this is referred to as the Pauli exclusion principle. Consequently, the general representation of a many-body state with identical fermions becomes

$$|\alpha_1\alpha_2\dots\alpha_N\rangle = \sqrt{N!}\,\mathcal{A}\,|\alpha_1\alpha_2\dots\alpha_N). \tag{2.11}$$

The phases contained in $\mathcal{A}$ is an important ingredient in this formula, as it ensures that the resulting many-body state is zero when at least two particles occupy the same quantum state. Since for instance

$$|\alpha_1\alpha_2\dots\alpha_N\rangle = -|\alpha_2\alpha_1\dots\alpha_N\rangle \tag{2.12}$$

represent the same physical state, the completeness relation should only consider one of these kets in the sum. This can be accomplished by the ordered sum

$$\sum_{\alpha_1\alpha_2\dots\alpha_N}^{\text{ordered}} |\alpha_1\alpha_2\dots\alpha_N\rangle\langle\alpha_1\alpha_2\dots\alpha_N| = 1. \tag{2.13}$$

If the single-particle quantum numbers are not ordered, the completeness relation instead reads

$$\frac{1}{N!}\sum_{\alpha_1\alpha_2\dots\alpha_N} |\alpha_1\alpha_2\dots\alpha_N\rangle\langle\alpha_1\alpha_2\dots\alpha_N| = 1, \tag{2.14}$$

because there are $N!$ equivalent states for each N-particle state.

A system of identical bosons is, in contrast to fermions, symmetric, and this implies that the total wavefunction of the system is unchanged when permuting two particles. For bosons there is therefore no exclusion principle, and all bosons can occupy the same quantum state. With these rules in mind, the formula for the state vector for bosons becomes

$$|\alpha_1\alpha_2\dots\alpha_N\rangle = \left[\frac{N!}{\prod_i n_i!}\right]^{1/2} \mathcal{S}\,|\alpha_1\alpha_2\dots\alpha_N), \tag{2.15}$$

where n_i is the multiplicity of state $|i\rangle$. One can conclude from this formula that e.g.

$$|\alpha_1\alpha_1\alpha_2\dots\alpha_N\rangle = |\alpha_1\alpha_2\alpha_1\dots\alpha_N\rangle, \tag{2.16}$$

and these are considered to be physically equivalent states. This implies that the completeness relation in Eq. (2.14) does not hold; instead one has to introduce weights $n_i!$ so that

$$\sum_{\alpha_1\alpha_2\dots\alpha_N} \frac{n_1!n_2!\dots}{N!}|\alpha_1\alpha_2\dots\alpha_N\rangle\langle\alpha_1\alpha_2\dots\alpha_N| = 1. \tag{2.17}$$

This is also equal to the sum in Eq. (2.13).

Example. (*Two identical particles*) In the case of two identical particles, the normalized state for fermions can be written as

$$|\alpha_1\alpha_2\rangle = \frac{1}{\sqrt{2}}\{|\alpha_1\alpha_2) - |\alpha_2\alpha_1)\}. \tag{2.18}$$

Note that if $\alpha_1 = \alpha_2$ the two-particle state is zero, due to the Pauli exclusion principle. For bosons the two-particle state becomes

$$|\alpha_1\alpha_2\rangle = \frac{1}{\sqrt{2n_1!n_2!}}\{|\alpha_1\alpha_2) + |\alpha_2\alpha_1)\}, \tag{2.19}$$

where $n_1 = 2$ and $n_2 = 0$ if $\alpha_1 = \alpha_2$. As expected the product states do not cancel even if $\alpha_1 = \alpha_2$. ■

Now that a general representation of symmetrized many-particle states has been established the next step is to find the corresponding wavefunctions in coordinate representation. The wavefunction of a many-body system can be written as a linear combination of product states of single-particle wavefunctions $\psi_\alpha(\boldsymbol{x}) = (\boldsymbol{x}|\alpha\rangle$. Incorporating the properties of antisymmetric states, the normalized wavefunction of identical fermions can be written as a so-called Slater determinant,

$$\psi_{\alpha_1\alpha_2\ldots\alpha_N}(\boldsymbol{x}_1\boldsymbol{x}_2\ldots\boldsymbol{x}_N) = \frac{1}{\sqrt{N!}}\det\begin{bmatrix}\psi_{\alpha_1}(\boldsymbol{x}_1) & \cdots & \psi_{\alpha_1}(\boldsymbol{x}_N)\\ \psi_{\alpha_2}(\boldsymbol{x}_1) & \cdots & \psi_{\alpha_2}(\boldsymbol{x}_N)\\ \vdots & \ddots & \vdots\\ \psi_{\alpha_N}(\boldsymbol{x}_1) & \cdots & \psi_{\alpha_N}(\boldsymbol{x}_N)\end{bmatrix}. \tag{2.20}$$

To formulate the corresponding wavefunction for bosons we first note that the *permanent* of a square, $n\times n$ matrix $A = (a_{ij})_{1\le i,j\le n}$ is defined by

$$\mathrm{perm}(A) = \sum_{\sigma\in S_n}\prod_{i=1}^{n} a_{i,\sigma(i)}, \tag{2.21}$$

where S_n is the set of permutations of $\{1,2,\ldots,n\}$. This is just the determinant without the minus signs, which in practice means that for a matrix $A = \begin{pmatrix}a & b\\ c & d\end{pmatrix}$, the permanent of A is

$$\mathrm{perm}\begin{pmatrix}a & b\\ c & d\end{pmatrix} = ad + bc. \tag{2.22}$$

The general wavefunction for an N-boson system is now

$$\psi_{\alpha_1\alpha_2\ldots\alpha_N}(\boldsymbol{x}_1\boldsymbol{x}_2\ldots\boldsymbol{x}_N) = \frac{1}{\sqrt{N!}}\,\mathrm{perm}\begin{bmatrix}\psi_{\alpha_1}(\boldsymbol{x}_1) & \cdots & \psi_{\alpha_1}(\boldsymbol{x}_N)\\ \psi_{\alpha_2}(\boldsymbol{x}_1) & \cdots & \psi_{\alpha_2}(\boldsymbol{x}_N)\\ \vdots & \ddots & \vdots\\ \psi_{\alpha_N}(\boldsymbol{x}_1) & \cdots & \psi_{\alpha_N}(\boldsymbol{x}_N)\end{bmatrix}. \tag{2.23}$$

2.2 Second quantization

The first quantization refers to the canonical quantization of classical theory, which leads to a semi-classical description of quantum mechanics. The disadvantage of this approach is that manipulating many-body systems becomes very difficult, and it can only be used for systems where the number of particles is fixed. A more suitable way to deal with many-particle systems is therefore to convert the operators to a second-quantized form and express them in terms of creation and annihilation operators. As it turns out, this will make many-body states much easier to handle. Second quantization is used in many fields of physics, and it is an essential part in e.g. quantum field theory.

2.2.1 Occupation number representation

Working with wavefunctions that are represented by Slater determinants and permanents of a many-body system may become tedious even for a small number of particles. Fortunately, it is possible to simplify many-body wavefunctions by the use of *occupation number representation*. Occupation number formalism can be used for both antisymmetric and symmetric multiparticle states. The idea is that the many-particle state $|\alpha_1\alpha_2\ldots\alpha_N\rangle$ is represented by specifying how many particles there are in each available state of the system. Thus the state is denoted as

$$|\alpha_1\alpha_2\ldots\alpha_N\rangle = |n_1 n_2 n_3\ldots\rangle, \tag{2.24}$$

where n_i is the number of particles in state $|i\rangle$. For a system of N particles we must have

$$\sum_{i=1}^{\infty} n_i = N. \tag{2.25}$$

If the particles in question are fermions then n_i can only be either zero or one, while for bosons it can be any non-negative integer as long as Eq. (2.25) holds. Table 2.1 lists a few of the occupation number bases for fermions and bosons. The states span the Hilbert spaces $\mathscr{H}_0$, $\mathscr{H}_1$, $\mathscr{H}_2$, ... The Hilbert space $\mathscr{H}_0$ has $N = 0$ and only contains the vacuum state $|0\rangle = |000...\rangle$, so it is customary to set $\mathscr{H}_0 = \mathbb{C}$. States with different number of particles are defined to be orthogonal.

Table 2.1. *Occupation number basis states that span the Hilbert spaces $\mathscr{H}_0$, $\mathscr{H}_1$, $\mathscr{H}_2$, ..., for both fermions and bosons.*

N	Occupation number basis states (fermions)
0	$\|0000...\rangle$
1	$\|1000...\rangle$, $\|0100...\rangle$, $\|0010...\rangle$, ...
2	$\|1100...\rangle$, $\|1010...\rangle$, $\|0110...\rangle$, $\|1001...\rangle$, $\|0101...\rangle$, $\|0011...\rangle$, ...
⋮	...
N	Occupation number basis states (bosons)
0	$\|0000...\rangle$
1	$\|1000...\rangle$, $\|0100...\rangle$, $\|0010...\rangle$, ...
2	$\|2000...\rangle$, $\|1100...\rangle$, $\|0200...\rangle$, $\|1010...\rangle$, $\|0110...\rangle$, $\|0020...\rangle$, $\|1001...\rangle$, ...
⋮	...

2.2.2 Fock space

So far we have only looked at Hilbert spaces that admit systems of a fixed number of particles. However, a more general treatment of many-body problems should allow for systems where the number of particles may vary. In the next section we will introduce operators that effectively changes the number of particles in a system, and this enforces us to employ the *Fock space*, denoted $\mathscr{F}$, which can handle systems with both varying and unknown number of particles.

There are different Fock spaces, depending on the particles that are involved and consequently what spaces the many-body states span. The *free* (or *full*) *Fock space* is spanned by occupation number bases and is an infinite direct sum of many-particle Hilbert spaces,

$$\mathscr{F} \equiv \mathscr{H}_0 \oplus \mathscr{H}_1 \oplus \mathscr{H}_2 \oplus \ldots = \bigoplus_{n=0}^{\infty} \mathscr{H}^{\otimes n}. \tag{2.26}$$

The states in this space may be fermionic, bosonic or an arbitrary combination of product states. There are also the *antisymmetric* and *symmetric Fock spaces*, corresponding to fermionic and bosonic systems, respectively. These Fock spaces use different tensor products and are constructed with the symmetric properties of the particles in mind.

2.2.3 Creation and annihilation operators

The creation operator $c_\alpha^\dagger$ and its Hermitian conjugate c_α, the annihilation operator, create and annihilate, respectively, a particle in state $|\alpha\rangle$. The creation operator is defined by

$$c_\alpha^\dagger|n_1 n_2 \dots n_\alpha \dots\rangle = \eta_\alpha^\dagger|n_1 n_2 \dots n_\alpha + 1 \dots\rangle, \tag{2.27}$$

where the phase factor

$$\eta_\alpha^\dagger = \begin{cases} \sqrt{1+n_\alpha} & \text{(bosons)} \\ \sqrt{1-n_\alpha}\,(-1)^{\sum_{\beta<\alpha} n_\beta} & \text{(fermions)} \end{cases} \tag{2.28}$$

For fermions the resulting state when applying the creation operator $c_\alpha^\dagger$ is thus zero if $n_\alpha = 1$, which it has to be because two fermions cannot occupy the same state. In all other cases the creation operator adds one particle to an N-particle state so that the system then contains $N+1$ particles. The action of the annihilation operator is

$$c_\alpha|n_1 n_2 \dots n_\alpha \dots\rangle = \begin{cases} \eta_\alpha|n_1 n_2 \dots n_\alpha - 1 \dots\rangle\,, & n_\alpha > 0 \\ 0\,, & n_\alpha = 0 \end{cases} \tag{2.29}$$

where the factor

$$\eta_\alpha = \begin{cases} \sqrt{n_\alpha} & \text{(bosons)} \\ \sqrt{n_\alpha}\,(-1)^{\sum_{\beta<\alpha} n_\beta} & \text{(fermions)} \end{cases} \tag{2.30}$$

There are a couple of operations that are important to know about when working with creation and annihilation operators. Important *commutation relations*[1] for bosons are

$$\begin{aligned} &[c_\alpha, c_\beta^\dagger] = \delta_{\alpha\beta} \\ &[c_\alpha, c_\beta] = [c_\alpha^\dagger, c_\beta^\dagger] = 0 \end{aligned} \qquad \text{(bosons)} \tag{2.31}$$

There is also the *anticommutation relations*[2] for fermions, given by

$$\begin{aligned} &\{c_\alpha, c_\beta^\dagger\} = \delta_{\alpha\beta} \\ &\{c_\alpha, c_\beta\} = \{c_\alpha^\dagger, c_\beta^\dagger\} = 0 \end{aligned} \qquad \text{(fermions)} \tag{2.32}$$

The vacuum state $|0\rangle$ can now be used in conjunction with creation operators in order to obtain an equivalent way of writing a many-particle state;

$$|\alpha_1 \alpha_2 \dots \alpha_N\rangle \equiv c_{\alpha_1}^\dagger c_{\alpha_2}^\dagger \dots c_{\alpha_N}^\dagger|0\rangle = \prod_i c_{\alpha_i}^\dagger|0\rangle. \tag{2.33}$$

[1] The commutator is defined as $[A,B] = AB - BA$.
[2] The anticommutator is defined as $\{A,B\} = AB + BA$.

However, this only holds for fermions. The anticommutation rules in Eq. (2.32) make sure that the state is antisymmetric, because

$$\begin{aligned} |\alpha_1\alpha_2\ldots\alpha_N\rangle &= c^\dagger_{\alpha_1}c^\dagger_{\alpha_2}\ldots c^\dagger_{\alpha_N}|0\rangle \\ &= \{c^\dagger_{\alpha_1}, c^\dagger_{\alpha_2}\}\ldots c^\dagger_{\alpha_N}|0\rangle - c^\dagger_{\alpha_2}c^\dagger_{\alpha_1}\ldots c^\dagger_{\alpha_N}|0\rangle \\ &= -|\alpha_2\alpha_1\ldots\alpha_N\rangle. \end{aligned} \tag{2.34}$$

Bosons will instead require a normalization factor when two or more particles occupy the same state. The general formula that holds for both fermions and bosons is

$$|n_{\alpha_1}n_{\alpha_2}\ldots\rangle = \prod_i \frac{1}{\sqrt{n_{\alpha_i}!}}(c^\dagger_{\alpha_i})^{n_{\alpha_i}}|0\rangle. \tag{2.35}$$

We have now found a way to easily construct many-body states from a system of no particles using creation operators, and the (anti)commutation rules enforce the states to be properly symmetrized.

2.2.4 Many-body operators

In order to proceed to the formulation of the equation of motion it is important to understand the action of operators on many-body states and to introduce the matrix representation of operators. We will now investigate how to quantize the relevant physical operators and express them in a more convenient form of creation and annihilation operators. This will make it easier to work with many-body systems.

The operators can be categorized according to how many particles they act on; there are *one-body operators*, *two-body operators* etc. In order to simulate interactions between particles that correspond to effective degrees of freedom it is usually necessary to include operators in the Hamiltonian that act on more than two particles at a time. Therefore, we will consider one-, two- and three-body operators in this work. Four-body operators and higher are rarely considered.

One-body operators

If $\hat{O}$ is a one-body operator in $\mathscr{H}_N$ it measures the quantum numbers of only one particle at a time. The action of $\hat{O}$ on an N-body state must therefore be given by the sum of its action on each particle, giving

$$\hat{O} = O_1 + O_2 + \ldots + O_N = \sum_{i=1}^{N} O_i \,, \tag{2.36}$$

where O_i acts solely on particle i. To better understand how O_i operates on a product state we may insert a complete set of states and write

$$\begin{aligned} O_i|\alpha_1\alpha_2...\alpha_N) &= |\alpha_1\rangle|\alpha_2\rangle...|\alpha_{i-1}\rangle\Big\{\sum_{\beta_i}|\beta_i\rangle\langle\beta_i|O|\alpha_i\rangle\Big\}|\alpha_{i+1}\rangle...|\alpha_N\rangle \\ &= \sum_{\beta_i}\langle\beta_i|O|\alpha_i\rangle|\alpha_1...\alpha_{i-1}\beta_i\alpha_{i+1}...\alpha_N), \end{aligned} \tag{2.37}$$

where the operator O acts in the single-particle space $\mathscr{H}$, and where we have used the completeness relation

$$\sum_{\beta_i}|\beta_i\rangle\langle\beta_i| = 1. \tag{2.38}$$

We see that the operator O_i picks particle i with quantum numbers α_i and the state is changed to $|\beta_i\rangle$ with probability $|\langle\beta_i|O|\alpha_i\rangle|^2$. If the matrix element $\langle\beta_i|O|\alpha_i\rangle$ is zero then of course this transition cannot happen. As the particles are identical the element $\langle\beta_i|O|\alpha_i\rangle$ only depends on the quantum numbers and not on which particle is considered. The total effect of $\hat{O}$ on a product state is

$$\begin{aligned} &\hat{O}|\alpha_1\alpha_2...\alpha_N) = O_1|\alpha_1\rangle|\alpha_2\rangle...|\alpha_N\rangle + \ldots + |\alpha_1\rangle|\alpha_2\rangle...O_N|\alpha_N\rangle \\ =&\sum_{\beta_1}\langle\beta_1|O|\alpha_1\rangle|\beta_1\alpha_2...\alpha_N) + \ldots + \sum_{\beta_N}\langle\beta_N|O|\alpha_N\rangle|\alpha_1\alpha_2...\beta_N) \\ =&\sum_{i=1}^{N}\sum_{\beta_i}\langle\beta_i|O|\alpha_i\rangle|\alpha_1...\alpha_{i-1}\beta_i\alpha_{i+1}...\alpha_N), \end{aligned} \tag{2.39}$$

by the use of Eqs. (2.36) and (2.37). The one-body operator $\hat{O}$ obviously commutes with both the antisymmetrizer $\mathcal{A}$ and the symmetrizer $\mathcal{S}$, that is

$$\begin{aligned} \hat{O}\mathcal{A} &= \mathcal{A}\hat{O}, \\ \hat{O}\mathcal{S} &= \mathcal{S}\hat{O}. \end{aligned} \tag{2.40}$$

Consequently, when the one-body operator $\hat{O}$ acts on an N-particle state of fermions one obtains

$$\begin{aligned} \hat{O}|\alpha_1\alpha_2\ldots\alpha_N\rangle &= \hat{O}\sqrt{N!}\,\mathcal{A}\,|\alpha_1\alpha_2...\alpha_N) = \sqrt{N!}\,\mathcal{A}\,\hat{O}|\alpha_1\alpha_2...\alpha_N) \\ &= \sqrt{N!}\,\mathcal{A}\sum_{i=1}^{N}\sum_{\beta_i}\langle\beta_i|O|\alpha_i\rangle|\alpha_1...\alpha_{i-1}\beta_i\alpha_{i+1}...\alpha_N) \\ &= \sum_{i=1}^{N}\sum_{\beta_i}\langle\beta_i|O|\alpha_i\rangle\sqrt{N!}\,\mathcal{A}\,|\alpha_1...\alpha_{i-1}\beta_i\alpha_{i+1}...\alpha_N) \\ &= \sum_{i=1}^{N}\sum_{\beta_i}\langle\beta_i|O|\alpha_i\rangle|\alpha_1...\alpha_{i-1}\beta_i\alpha_{i+1}...\alpha_N\rangle, \end{aligned} \tag{2.41}$$

and a similar treatment for bosons yields the same result. Hence, no matter if the particles in a system are fermions or bosons we have

$$\hat{O}|\alpha_1\alpha_2\ldots\alpha_N\rangle = \sum_{i=1}^{N}\sum_{\beta_i}\langle\beta_i|O|\alpha_i\rangle|\alpha_1\ldots\alpha_{i-1}\beta_i\alpha_{i+1}\ldots\alpha_N\rangle. \tag{2.42}$$

Considering how $\hat{O}$ operates on product and many-body states we may write it in the matrix representation

$$\hat{O} = \sum_{\alpha,\beta}|\beta\rangle\langle\beta|O|\alpha\rangle\langle\alpha|. \tag{2.43}$$

The simple interpretation of this is again that in each term the particle with state $|\alpha\rangle$ is picked and transformed into $|\beta\rangle$ with probability $|\langle\beta|O|\alpha\rangle|^2$. The sum over α is in accordance with the expression of $\hat{O}$ in Eq. (2.36).

The next step is to generalize the one-body operator in Eq. (2.43) by expressing it in terms of creation and annihilation operators. This will yield an operator $\hat{O}$ in Fock space. It can be done by using the fact that $\langle\alpha| = (c_\alpha^\dagger|0\rangle)^\dagger = \langle 0|c_\alpha$ and $|\beta\rangle = c_\beta^\dagger|0\rangle$, and the result is

$$\hat{O} = \sum_{\alpha,\beta}\langle\beta|O|\alpha\rangle c_\beta^\dagger|0\rangle\langle 0|c_\alpha = \sum_{\alpha,\beta}\langle\beta|O|\alpha\rangle c_\beta^\dagger c_\alpha. \tag{2.44}$$

We see that the annihilation operator c_α destroys the particle of state $|\alpha\rangle$, and the operator $c_\beta^\dagger$ then creates a particle in state $|\beta\rangle$. The matrix element $\langle\beta|O|\alpha\rangle$ is the amplitude of this transition. The one-body operator is now expressed in terms of creation and annihilation operators and this was the result we aimed for. It is much easier to use this form in practice, because the problem is now reduced to calculating the matrix elements $\langle\beta|O|\alpha\rangle$.

Example. (*The number operator*) An interesting example of a one-body operator is

$$\hat{N} = \sum_{\alpha} c_\alpha^\dagger c_\alpha\ . \tag{2.45}$$

When this operator acts on a many-particle state one obtains, by using Eq. (2.44),

$$\begin{aligned}\hat{N}|\alpha_1\alpha_2...\alpha_N\rangle &= \sum_{i,j}\langle\beta_j|c_{\alpha_i}^\dagger c_{\alpha_i}|\alpha_i\rangle c_{\beta_j}^\dagger c_{\alpha_i}|\alpha_1\alpha_2...\alpha_N\rangle \\ &= \sum_{i,j}\langle\beta_j|\alpha_i\rangle|\alpha_1...\alpha_{i-1}\beta_i\alpha_{i+1}...\alpha_N\rangle \\ &= \sum_{i}|\alpha_1...\alpha_{i-1}\alpha_i\alpha_{i+1}...\alpha_N\rangle = N|\alpha_1\alpha_2...\alpha_N\rangle,\end{aligned} \tag{2.46}$$

so this operator counts how many particles a particular system contains. It is therefore often called the number operator, or the occupation number operator. ■

Other important one-body operators is the one of kinetic energy, $\hat{T}$, and the angular momentum operators $\hat{J}_z$ and $\hat{J}_\pm$. The eigenvalues of $\hat{T}$ are the kinetic energies of the particles.

Many-body operators

Now that we understand one-body operators it is easier to extend the formalism to many-body operators. Two-body operators involve two interacting particles and operate on every pair of particles in a system. Since the particles are identical it does not matter if two particles in a pair are interchanged, and one should make sure that every pair is counted only once. If the operator $\hat{V}$ is a two-body operator, its total effect on a system of N particles is the sum of its action on each pair of particles,

$$\begin{aligned}\hat{V} = V_{1,2} + V_{1,3} + \ldots + V_{1,N} + \\ V_{2,3} + \ldots + V_{2,N} + \\ \ddots \quad \vdots \\ V_{N-1,N} \end{aligned} = \sum_{i<j=1}^{N} V_{i,j} = \tfrac{1}{2}\sum_{i\neq j}^{N} V_{i,j} \ , \tag{2.47}$$

where $V_{i,j}$ acts on particles i and j. Note that $V_{i,j} = V_{j,i}$ because of the indistinguishability of the particles. If $V_{i,j}$ acts on a product state a similar derivation as in Eq. (2.37) for the one-body operator yields

$$\begin{aligned}&V_{i,j}|\alpha_1\alpha_2\ldots\alpha_N) = \\ &\quad\sum_{\beta_i\beta_j}(\beta_i\beta_j|V|\alpha_i\alpha_j)|\alpha_1\ldots\alpha_{i-1}\beta_i\alpha_{i+1}\ldots\alpha_{j-1}\beta_j\alpha_{j+1}\ldots\alpha_N\rangle,\end{aligned} \tag{2.48}$$

where V operates in $\mathscr{H}_2$. The total effect of $\hat{V}$ on the product state is thus

$$\hat{V}|\alpha_1\alpha_2\ldots\alpha_N) = \sum_{i<j=1}^{N}\sum_{\beta_i\beta_j}(\beta_i\beta_j|V|\alpha_i\alpha_j)|\alpha_1\ldots\beta_i\ldots\beta_j\ldots\alpha_N). \tag{2.49}$$

The operator $\hat{V}$ does like the one-body operator commute with both the antisymmetrizer $\mathcal{A}$ and the symmetrizer $\mathcal{S}$, and using the same approach as before will finally yield the action of $\hat{V}$ on a many-body state,

$$\hat{V}|\alpha_1\alpha_2\ldots\alpha_N\rangle = \sum_{i<j=1}^{N}\sum_{\beta_i\beta_j}(\beta_i\beta_j|V|\alpha_i\alpha_j)|\alpha_1\ldots\beta_i\ldots\beta_j\ldots\alpha_N\rangle. \tag{2.50}$$

$\hat{V}$ can also be written in the concise form

$$\hat{V} = \sum_{\alpha\beta} \sum_{\gamma\delta} |\alpha\beta)(\alpha\beta|V|\gamma\delta)(\gamma\delta|, \tag{2.51}$$

and using creation and annihilation operators one obtains

$$\hat{V} = \frac{1}{2} \sum_{\alpha\beta\gamma\delta} (\alpha\beta|V|\gamma\delta) c_\alpha^\dagger c_\beta^\dagger c_\delta c_\gamma \ . \tag{2.52}$$

This holds for both fermions and bosons. It is possible to extend this to operators that work on an arbitrary number of particles; an m-body operator would be written as

$$\hat{M} = \frac{1}{m!} \sum_{12...m} \sum_{1'2'...m'} (12...m|M|1'2'...m') c_1^\dagger c_2^\dagger ... c_m^\dagger c_{m'} c_{(m-1)'} ... c_{1'} \ . \tag{2.53}$$

Finally we can express the Hamiltonian in a second quantized form with one- and two-body operators. If $\hat{T}$ and $\hat{V}$ are the operators of kinetic and potential energy, respectively, the Hamiltonian may be written as

$$\hat{H} = \hat{T} + \hat{V} = \sum_{\alpha\beta} \langle\alpha|T|\beta\rangle c_\alpha^\dagger c_\beta + \frac{1}{2} \sum_{\alpha\beta\gamma\delta} (\alpha\beta|V|\gamma\delta) c_\alpha^\dagger c_\beta^\dagger c_\delta c_\gamma \ , \tag{2.54}$$

and of course this must hold for both fermions and bosons. Including three-body forces would result in

$$\begin{aligned} \hat{H} = &\sum_{\alpha\beta} \langle\alpha|T|\beta\rangle c_\alpha^\dagger c_\beta + \frac{1}{2} \sum_{\alpha\beta\gamma\delta} (\alpha\beta|V|\gamma\delta) c_\alpha^\dagger c_\beta^\dagger c_\delta c_\gamma \\ &+ \frac{1}{6} \sum_{\alpha\beta\gamma} \sum_{\alpha'\beta'\gamma'} (\alpha\beta\gamma|W|\alpha'\beta'\gamma') c_\alpha^\dagger c_\beta^\dagger c_\gamma^\dagger c_{\gamma'} c_{\beta'} c_{\alpha'} \ . \end{aligned} \tag{2.55}$$

The sums are over all possible single-particle states so there is no assumption on how many particles there are in the system. If $|i\rangle$ and $|f\rangle$ are the initial and final states, the matrix elements become $\langle f|\hat{H}|i\rangle$. The calculation of the matrix elements is now simplified by the use of the formulation of $\hat{H}$ in Eqs. (2.54) and (2.55) and the commutation and anticommutation rules given in Eqs. (2.31) and (2.32).

2.3 Coupled and uncoupled schemes

A coupled quantum state involves two or more particles that are linked in a way that changing the state of one particle affects the others as well. It is similar to entangled states, but entanglement often refers to particles that are also correlated over relatively large distances.

Starting with two particles with total angular momentum j_1 and j_2, with projections m_1 and m_2 that can have values

$$m_1 = -j_1, -j_1+1, \ldots, j_1-1, j_1,$$
$$m_2 = -j_2, -j_2+1, \ldots, j_2-1, j_2,$$

their single-particle states are $|j_1 m_1\rangle$ and $|j_2 m_2\rangle$.

Now, in order to construct a two-particle state with these two particles, there are two choices. The uncoupled two-body state is the product state $|j_1 m_1\rangle|j_2 m_2\rangle$ and the coupled two-body state, expanded in uncoupled states, is given by

$$|j_{12} m_{12}\rangle = \sum_{m_1}\sum_{m_2} C^{j_{12} m_{12}}_{j_1 m_1 j_2 m_2} |j_1 m_1\rangle|j_2 m_2\rangle, \tag{2.56}$$

where C are the Clebsch-Gordan coefficients. The coefficients are zero if either $|j_1 - j_2| \leq j_{12} \leq |j_1 + j_2|$ or $m_{12} = m_1 + m_2$ does not hold. These conditions are important for coupled states. The normalization condition for the Clebsch-Gordan coefficients is given by

$$\sum_{m_1}\sum_{m_2} \left(C^{j_{12} m_{12}}_{j_1 m_1 j_2 m_2}\right)^2 = 1. \tag{2.57}$$

Example. (*Two coupled spin-$\frac{1}{2}$ particles*) In the case of two spin-$\frac{1}{2}$ particles their projections are $m_{s_1} = \pm\frac{1}{2}$ and $m_{s_2} = \pm\frac{1}{2}$. The particles are fermions because they have half-integer spin. The coupled state is given by

$$|s_{12} m_{s_{12}}\rangle = \sum_{m_{s_1}}\sum_{m_{s_2}} C^{s_{12} m_{s_{12}}}_{s_1 m_{s_1} s_2 m_{s_2}} |s_1 m_{s_1}\rangle|s_2 m_{s_2}\rangle. \tag{2.58}$$

The possible values for the total spin $s_{12} = \frac{1}{2} \pm \frac{1}{2} = 0, 1$ and the projection is $m_{s_{12}} = -1, 0, 1$ if $s_{12} = 1$ and $m_{s_{12}} = 0$ if $s_{12} = 0$. Thus different states are possible, given by

$$\begin{aligned}
|11\rangle &= |\tfrac{1}{2}\tfrac{1}{2}\rangle|\tfrac{1}{2}\tfrac{1}{2}\rangle \\
|10\rangle &= \frac{1}{\sqrt{2}}\left(|\tfrac{1}{2}\tfrac{1}{2}\rangle|\tfrac{1}{2}{-}\tfrac{1}{2}\rangle + |\tfrac{1}{2}{-}\tfrac{1}{2}\rangle|\tfrac{1}{2}\tfrac{1}{2}\rangle\right) \\
|1{-}1\rangle &= |\tfrac{1}{2}{-}\tfrac{1}{2}\rangle|\tfrac{1}{2}{-}\tfrac{1}{2}\rangle \\
|00\rangle &= \frac{1}{\sqrt{2}}\left(|\tfrac{1}{2}\tfrac{1}{2}\rangle|\tfrac{1}{2}{-}\tfrac{1}{2}\rangle - |\tfrac{1}{2}{-}\tfrac{1}{2}\rangle|\tfrac{1}{2}\tfrac{1}{2}\rangle\right)
\end{aligned} \tag{2.59}$$

Note that the Clebsch-Gordan coefficients satisfy the normalization condition in Eq. (2.57). ■

One may extend this to more than two particles. If three particles are considered several coupling schemes are possible. The coupled state may for instance be written as

$$|j_{123}m_{123}\rangle = \sum_{\substack{m_{12}\\m_3}} C^{j_{123}m_{123}}_{j_{12}m_{12}j_3m_3}|j_{12}m_{12}\rangle|j_3m_3\rangle = \sum_{\substack{m_{12}\\m_3}}\sum_{\substack{m_1\\m_2}} C^{j_{123}m_{123}}_{j_{12}m_{12}j_3m_3}C^{j_{12}m_{12}}_{j_1m_1j_2m_2}|j_1m_1\rangle|j_2m_2\rangle|j_3m_3\rangle. \tag{2.60}$$

The quantum numbers j_{123} and m_{123} should satisfy $j_{123} \geq |m_{123}|$ and $m_{123} = m_1 + m_2 + m_3$.

There are a couple of advantages and disadvantages using either coupled or uncoupled schemes. The coupled scheme has a smaller basis dimension, which means that the Hamiltonian matrix will be smaller. On the other hand, it is very complicated to evaluate the matrix elements, and the Hamiltonian is less sparse. In contrast, using uncoupled schemes, like the m-scheme, demands a larger dimension of the many-body basis to obtain an equivalent approximation. However, the resulting matrix contains less nonzero values, and calculating and storing matrix elements is easy. A more detailed account regarding this is found in [6].

3 No-Core Shell Model

Many-body physics can be distinguished from *few-body* physics by the number of particles that is considered. A few-body system may contain three or four particles, whereas many-particle systems do not have any such restriction. The distinction is induced by the fact that some basically exact numerical techniques, such as solving the Faddeev equations, exist for three-body systems, but not for larger number of particles. At the moment there are few approaches available that successfully solve the many-body problem, with $N > 4$, with a realistic potential [7]. This chapter will briefly describe one of these approaches, called the No-Core Shell Model (NCSM), by employing the tools and ideas discussed in the previous chapter.

The NCSM was developed in nuclear physics to study systems with strongly interacting fermionic nucleons. The dimensionality problem of the many-body system is handled by utilizing a truncated model space, and this makes it possible to generate a Hamiltonian matrix of finite size. Because of the finite model space a truncated harmonic oscillator (HO) basis is used in NCSM [8]. However, a consequence of using a truncated basis is that strong short-range correlations cannot be accounted for. It is therefore necessary to construct *effective interactions*, which turn to the original realistic potentials as the size of the basis tends to infinity.

In this chapter the modified, effective Hamiltonian and the truncated HO basis used in NCSM are first introduced. Some characteristics of the matrix elements will then be discussed. A knowledge of Jacobi coordinates may be in order when reading these sections, and this is covered in Appendix A. Lastly, in Section 3.4 the No-Core Shell Model for bosons, or NCSMb for short, is introduced. This computer code was the centerpiece of our study. Specifically, the essential functions of the subroutine that computes the matrix elements for an arbitrary number of bosons, will be investigated to get a better picture of how the different aspects of the many-body theory come together. Note that, as opposed to the usual systems studied with the NCSM, NCSMb does not work with fermions and thus the particles cannot be nucleons. The systems may instead consist of bosonic atoms such as ^{4}He.

3.1 The Hamiltonian

The Hamiltonian for the N-body system, denoted by $\hat{H}$, should include the kinetic energy of the particles and the interactions between them. We will consider realistic two- as well as three-body potentials that have been derived using theory together with experimental data. Denote the two-body potential $\hat{V} = \sum_{i<j} V_{i,j}$ and the three-body interaction $\hat{W} = \sum_{i<j<k} W_{i,j,k}$. To start with we express the Hamiltonian as

$$\hat{H}_{\text{rel}} = \sum_{i<j}^{N} \frac{(\hat{p}_i - \hat{p}_j)^2}{2m} + \sum_{i<j}^{N} V_{i,j} + \sum_{i<j<k}^{N} W_{i,j,k} \, . \tag{3.1}$$

The NCSMb can handle particles in an external trapping potential. The trap is modelled by an HO potential

$$\begin{aligned} \hat{H}_{\text{HO}}(\Omega) &= \hat{T} + \frac{1}{2} N m \Omega^2 \hat{R}^2 = \hat{T} + \frac{1}{2} N m \Omega^2 \left(\frac{1}{N} \sum_{i=1}^{N} \hat{r}_i \right)^2 \\ &= \sum_{i=1}^{N} \frac{\hat{p}_i}{2m} + \sum_{i=1}^{N} \frac{1}{2} m \Omega^2 \hat{r}_i^2 - \sum_{i<j}^{N} \frac{m\Omega^2}{2N} (\hat{r}_i - \hat{r}_j)^2. \end{aligned} \tag{3.2}$$

Combining this with $\hat{H}$ finally gives the total Hamiltonian

$$\begin{aligned} \hat{H}_{\text{tot},\Omega} = \hat{H}_{\text{rel}} + \hat{H}_{\text{HO}}(\Omega) = \sum_{i=1}^{N} \left[\frac{\hat{p}_i}{2m} + \frac{1}{2} m \Omega^2 \hat{r}_i^2 \right] + \\ \sum_{i<j}^{N} \left[V_{i,j} - \frac{m\Omega^2}{2N} (\hat{r}_i - \hat{r}_j)^2 \right] + \sum_{i<j<k}^{N} W_{i,j,k}, \end{aligned} \tag{3.3}$$

where the terms are grouped into one-, two-, and three-body operators.

3.2 Harmonic oscillator basis

The single-particle basis used in the NCSM is the harmonic oscillator basis. Using a total energy cut-off it allows translational invariance also for truncated model spaces. The basis is constructed by the analytical solutions of particles in a HO potential, when the correlations between the particles are not taken into account. The wavefunction of a HO state $|\alpha\rangle$ with quantum numbers n, l and m can be written in the form

$$(\boldsymbol{x}|\alpha\rangle = (\boldsymbol{x}|nlm\rangle = \psi_{nlm}(\boldsymbol{x}) = R_{nl}(r) Y_{lm}(\theta, \varphi), \tag{3.4}$$

where the radial wavefunction is given by

$$R_{nl} = \left[\frac{(2n)!!2^{l+2}}{(2(n+l)+1)!!\sqrt{\pi}}\right]^{1/2} r^l e^{-r^2/2} L_n^{l+1/2}(r^2). \tag{3.5}$$

In the HO basis all lengths are measured in units of $b = \sqrt{\hbar/m\Omega}$, which is related to the HO frequency Ω. The letters m will in this chapter interchangeably be used for both mass quantity and the magnetic quantum number. The wavefunctions are orthonormal and satisfy

$$\int \mathrm{d}^3x\ \psi^\dagger_{n'l'm'}(\boldsymbol{x})\psi_{nlm}(\boldsymbol{x}) = \delta_{nn'}\delta_{ll'}\delta_{mm'}. \tag{3.6}$$

The eigenstates of a single particle in an HO potential are given by

$$\hat{H}_{\mathrm{HO}}\,\psi_{nlm}(\boldsymbol{x}) = \hbar\omega\left(2n+l+\tfrac{3}{2}\right)\,\psi_{nlm}(\boldsymbol{x}) \equiv \hbar\omega\left(N+\tfrac{3}{2}\right)\,\psi_{nlm}(\boldsymbol{x}). \tag{3.7}$$

From now on, if not explicitly stated otherwise, the energies will be expressed in units of $\hbar\omega$ and the zero-energy, the 3/2-term, is omitted. The energy is thus $N = 2n + l$, and since the parity is given by $\pi = (-1)^l$, clearly

$$\pi = (-1)^l = (-1)^{N-2n} = (-1)^N, \tag{3.8}$$

which means that the parity is completely determined by N.

As have already been stated, the NCSM works in a finite model space defined by a maximum allowed energy for the N particles; denote this cut-off energy $N_{\max}$. This means that the sum of all energies of the particles cannot exceed this parameter, that is

$$\sum_{i=1}^{N} N_i \leq N_{\max}\,, \tag{3.9}$$

where N is the number of particles and N_i their energies. This conflicting notation is unfortunate, but hopefully it will always be clear what quantity N refers to.

3.3 Calculation of matrix elements

Using the basis expansion of the eigenvectors the Schrödinger equation turns into a matrix eigenvalue problem. We now turn to the calculation of matrix elements from the Hamiltonian operator written in the second-quantized form

$$\begin{aligned}\hat{H} =& \sum_{\alpha\beta}\langle\alpha|T|\beta\rangle c^\dagger_\alpha c_\beta + \frac{1}{2}\sum_{\alpha\beta\gamma\delta}(\alpha\beta|V|\gamma\delta)c^\dagger_\alpha c^\dagger_\beta c_\delta c_\gamma \\ &+ \frac{1}{6}\sum_{\alpha\beta\gamma}\sum_{\alpha'\beta'\gamma'}(\alpha\beta\gamma|W|\alpha'\beta'\gamma')c^\dagger_\alpha c^\dagger_\beta c^\dagger_\gamma c_{\gamma'}c_{\beta'}c_{\alpha'}\ .\end{aligned} \tag{3.10}$$

The matrix elements are then given by $\langle f|\hat{H}|i\rangle$, as stated before. It is easy to see, using occupation number representation, that an a-body operator only gives a nonzero contribution to the matrix element if the initial (i) and the final (f) many-body basis states differ from each other with at most a single-particle states. Take for instance the two-body operator $\hat{V}$, and initial and final states $|n_1'n_2'n_3'\ldots\rangle$ and $|n_1n_2n_3\ldots\rangle$, respectively. A term in the sum over the single-particle states is then given by

$$\begin{aligned}\langle n_1n_2n_3\ldots|(\alpha\beta|V|\gamma\delta)c_\alpha^\dagger c_\beta^\dagger c_\delta c_\gamma|n_1'n_2'n_3'\ldots\rangle = \\ = \eta_\alpha^\dagger\eta_\beta^\dagger\eta_\gamma\eta_\delta(\alpha\beta|V|\gamma\delta) \\ \times\langle n_1n_2n_3...|n_1'n_2'...n_\alpha'{-}1...n_\beta'{-}1...n_\gamma'{+}1...n_\delta'{+}1...\rangle,\end{aligned} \tag{3.11}$$

where the factors $\eta_\alpha^\dagger$ and η_α result from the action of the creation and annihilation operators and are given in Eqs. (2.28) and (2.30). This term is nonzero if the coefficients are nonzero and if

$$|n_1n_2n_3\ldots\rangle = |n_1'n_2'...n_\alpha'{-}1...n_\beta'{-}1...n_\gamma'{+}1...n_\delta'{+}1...\rangle,$$

because the states are orthogonal. If more than two states differ this equation does not hold for any (α,β,γ,δ). It is therefore not surprising that a Hamiltonian with three-body operators will result in a matrix with less zeros than obtained with only one- and two-body operators.

There are different methods of calculating the two-body matrix elements $(\alpha\beta|V|\gamma\delta)$, and one can use either coupled or uncoupled schemes. We will not delve deeply into it except giving some ideas on how it is done in NCSMb in the next section.

3.4 No-Core Shell Model for bosons

The No-Core Shell Model for bosons uses the same techniques as NCSM but some modifications are made to accommodate for symmetric particles. The code is written by Simon Tölle from the Bonn University in collaboration with C. Forssén, H. T. Johansson, and L. Platter at Chalmers University of Technology. What follows is a summary of the first few tasks performed by the NCSMb code. It is by no means an extensive description, but relevant features are discussed.

Create single-particle basis

The first thing the program does is to create the single-particle basis for a model space with a cut-off parameter $N_{\max}$ and with the number of bosons given by N_{boson}. This is done by letting the energy $N = 2n + l$ be iterated over all integers from zero to $N_{\max}$. For every value of l there are $2l + 1$ values of m given by

$$m = -l, -l{+}1, \ldots, l{-}1, l\ . \tag{3.12}$$

The single-particle states are denoted $|nlm\rangle$ and are completely determined by these quantum numbers. However, apart from satisfying the condition in Eq. (3.9) there is also a condition on the m_i quantum numbers, given by

$$M = \sum_{i=1}^{N_{\text{boson}}} m_i\ . \tag{3.13}$$

All single-particle states that fulfill these conditions are used to make up the many-particle states.

Example. Let $N_{\max} = 2$, $N_{\text{boson}} = 3$ and $M = 0$. N should then go from zero to two.

$\underline{N = 0}$**:** n and l must be zero and consequently $m = 0$.

$\underline{N = 1}$**:** The only possible values for n and l are $n = 0$ and $l = 1$. Thus $m = -1, 0, 1$.

$\underline{N = 2}$**:** Either $n = 0$ and $l = 2$, or $n = 1$ and $l = 0$. In the first case $m = -2, -1, 0, 1, 2$ and in the second case $m = 0$.

The many-body basis (see below) will be composed of combinations of these single-particle states. However, the total-M condition adds an additional constraint and there will be no configuration of the three particles that contains the states $|nlm\rangle = |02{-}2\rangle$, $|02{-}1\rangle$, $|021\rangle$ or $|022\rangle$. The single-particle basis is therefore made up of the states

$$|000\rangle, \quad |01{-}1\rangle, \quad |010\rangle, \quad |011\rangle, \quad |100\rangle, \quad |020\rangle$$

■

Create two- and three-particle bases

The next step of the computer program is to create a two- and three-particle basis, because this will facilitate the computation of two- and three-body matrix elements. It is not necessary to construct a three-particle basis if three-body forces are not used. The way these bases are constructed is by combining the quantum numbers of the individual particles.

Consider the case of two particles and denote $N_1 = 2n_1+l_1$ and $N_2 = 2n_2+l_2$. These combine into $N_1 + N_2 = N$, which should be less than or equal to the energy cut-off $N_{\max}$. Furthermore, the total angular momentum L is in the range

$$|l_1 - l_2| \leq L \leq l_1 + l_2,$$

and $M = m_1 + m_2$, where the magnetic quantum numbers m_i are obtained from the angular momenta l_i as in Eq. (3.12). The basis consists of coupled two-particle states $|(n_1 l_1 n_2 l_2)LM\rangle$ that are constructed by all possible combinations of the single-particle states. In the code NCSMb, an array of indices is created in order to easily map from two single-particle states to their corresponding two-particle states, and this is used when two-particle interactions are considered.

Create many-particle basis

The size of the many-body basis corresponds to the dimension of the matrix. The many-particle states are formed by all possible combinations of single-particle states, each having the same value of M, total energy $N \leq N_{\max}$, and total parity Π.

The many-particle states are represented by occupation numbers, giving states $|n_1 n_2 n_3 \ldots\rangle$, because this makes it easier to remove and add single-particle states. Since the single-particle states are referenced by indices it is, by the use of occupation number representation, very easy to check if a specific state is occupied for a specific multiparticle state. To facilitate the search through many-particle states a lookup table is made in NCSMb, and each state is referenced by a number.

Compute matrix elements

The calculation of many-body matrix elements is a rather complicated task that involves a lot of bookkeeping. In short one has to find all pairs of many-body basis states that differ by at most two (three) single-particle states when studying contributions from two-(three-)body operators. The two- and three-body matrix elements that are needed to evaluate the many-body

matrix elements, see Eq. (3.11), can either be read from file or computed assuming contact interactions between the particles

$$\hat{V} = c\delta^3(\boldsymbol{x}_1 - \boldsymbol{x}_2). \tag{3.14}$$

An efficient algorithm for computing the many-body matrix involves fast identification of the location of nonzero matrix elements. A subroutine in NCSMb loops over initial states $|I\rangle$ in the many-body basis and finds all possible final states $|F\rangle$ that can be reached by two-body jumps. A many-body state is stored as an array with N entries specifying the occupied single-particle states. In summary, the algorithm includes the following steps:

1. Loop over all many-body states in the model basis, and in each iteration use the multiparticle state as an initial state I.
2. Within the outer loop, loop over pairs of particle indices $\sum_{i=1}^{N}\sum_{j>i}^{N}$. This should correspond to single-particle indices $\alpha_i\alpha_j$ in the initial state I.
3. For each pair ij; remove $\alpha_i\alpha_j$ and create all possible (see below) new pairs of single-particle states $\beta_i\beta_j$ at this position. For each such replacement we will form a final state F.
4. In general, the state $|I - \alpha_i\alpha_j + \beta_i\beta_j\rangle$ does not have the required property of ascending single particle indices. Thus, the state must be sorted to accomplish the next step.
5. Figure out the index of this final state (using a hash table) and compute the contribution to the many-body matrix element $V_{IF}(ij,\beta_i\beta_j)$.

Diagonalize the matrix

When the Hamiltonian matrix has been generated, the next step is to diagonalize the matrix to find the eigenenergies and eigenstates. Only the smallest eigenvalues are generally sought, which includes the ground state energy and the first excitation energies. A suitable candidate for the diagonalization of the Hamiltonian is the Lanczos method, which is an efficient method to apply on symmetric, sparse matrices.

Moreover, the first eigenvalues that converge using Lanczos are the least and the largest which, in terms of execution time, is advantageous in this case. The Lanczos method and its variations is therefore the topic of the following chapter.

4 Numerical Methods

The previous chapter described the elementary theory of quantum mechanical many-body problems involving systems of identical, interacting bosons or fermions. An expansion in terms of a many-body basis led to a matrix formulation of the Hamiltonian eigenvalue problem. These matrices are symmetric, sparse and generally of large order. Furthermore, physicists are often only interested in finding the energy of the ground state and possibly of a few of the first excited states. For these reasons, traditional eigensolvers based on QR-factorization and similar techniques are ill-fitted for the task at hand. The main disadvantages are:

1. An excessive amount of storage space is required.
2. Unwanted eigenpairs are computed.
3. Many methods are based on similarity transformations that destroy the sparse structure.
4. Convergence requires gargantuan amounts of arithmetic operations.

Fortunately, there is family of methods much better suited for dealing with large, sparse and symmetric eigenvalue problems, namely methods based on the *Lanczos algorithm*. Originally conceived by Cornelius Lanczos[1] in 1950, the early versions of the Lanczos algorithm were used to solve an eigenvalue problem by computing what is known as a minimal polynomial. This approach however led to issues regarding implementation and Lanczos' work was quickly overshadowed by methods that were considered superior, such as QR-iteration. It was not until 20 years later that the true potential of the Lanczos algorithm was discovered by the pioneering work of C. Paige and G. Golub, among others. These authors suggested the use of the Lanczos algorithm as a means of solving large and sparse eigenproblems and they are largely responsible for the modern interpretation of the Lanczos algorithm given in this chapter. Another notable application of Lanczos-based methods is in the field of structural mechanics, where numerical solutions based on the method of finite elements lead to generalized sparse eigenproblems.

The remainder of this chapter introduces several important topics regarding the Lanczos algorithm. Section 4.1 contains a general introduction to

[1] Lanczos was notably an assistant to Einstein and made several important contributions in the fields of mathematics and theoretical physics.

Krylov Subspace Methods which is a much larger family of methods of which the Lanczos-based methods are only a subset. The next section describes how to extend the Lanczos algorithm to operate on blocks, appropriately named the Block Lanczos algorithm. The last few sections explores modifications of these algorithms to accommodate for finite precision and memory requirements.

4.1 Krylov subspace methods

In this section the basics of the Krylov Space is presented, which in turn leads to the Lanczos algorithm. The basic advantage of the Lanczos algorithm is that it makes it possible to by only a few matrix multiplications create a new matrix of much smaller dimension than the original, containing information about the extreme eigenvalues and eigenvectors of the original matrix. The structure and content of this section is largely based on a similar exposition found in [9].

Assume A is an symmetric $n \times n$ matrix with eigenvalues $(\lambda_i)_{i=1}^n$ and associated eigenvectors $(v_i)_{i=1}^n$. An arbitrary real vector x_0 of length n can then be expanded in the base of the eigenvectors

$$x_0 = \sum_{i=1}^{n} c_{i,0}\, v_i, \tag{4.1}$$

where $c_{i,0}$ are real coefficients. After multiplying with the matrix A the result will therefore become

$$Ax_0 = \sum_{i=1}^{n} \lambda_i c_{i,0} v_i = \sum_{i=1}^{n} c_{i,1} v_i, \tag{4.2}$$

where $c_{i,1} = \lambda_i c_{i,0}$. Observe that if two eigenvalues are distinct, $\lambda_i \neq \lambda_j$, then

$$\frac{c_{i,1}}{c_{i,0}} \neq \frac{c_{j,1}}{c_{j,0}}, \tag{4.3}$$

thus making the two vectors x_0 and Ax_0 linearly independent.

In fact, it can be shown that the whole sequence $(A^{k-1}x_0)_{k=1}^{n'}$ is linearly independent, where n' is the number of distinct eigenvalues λ_i for which there is a starting coefficient $c_{i,0} \neq 0$. Any vector $A^{n'}x_0$ will at the same time be possible to express with the earlier ones, and because $n' \leq n$, that will always be true for $A^n x_0$ as well.

The sequence $(A^{k-1}x_0)_{k=1}^{n'}$ is called a Krylov sequence and the matrix

$$K_k = [x_0 \; Ax_0 \; \ldots \; A^{k-1}x_0] = [x_0 \; x_1 \; \ldots \; x_{k-1}] \tag{4.4}$$

is defined as the $n \times k$ Krylov matrix, with the Krylov subspace $\mathcal{K}_k = \text{span}(K_k)$. If K_k is multiplied by A, then

$$\begin{aligned} &AK_n = [Ax_0 \; Ax_1 \; \ldots \; Ax_{n-1}] = [x_1 \; x_2 \; \ldots \; x_n] = \\ &K_n[e_2 \; e_3 \; \ldots \; e_n \; K_n^{-1}x_n] \equiv K_n C_n, \end{aligned} \tag{4.5}$$

assuming $n' = k = n$, and that K_n is not singular. e_i denotes the ith unit vector. Notice that C_n lacks nonzero elements below its first sub-diagonal. Matrices of that form are defined as *upper Hessenberg*. Furthermore, the relation

$$AK_n = K_n C_n \tag{4.6}$$

means that A is *similar* to C_n, which in turn means that if A is symmetric, then so is C_n. Considering that C_n is both upper Hessenberg and symmetric, all of the nonzero elements of C_n needs to be either on its diagonal or its first sub-diagonal. C_n is thus a *tridiagonal* matrix.

K_n on the other hand is a basis for $\mathcal{K}_n$. There is however an issue associated with this basis. Since

$$x_k = A^k x_0 = \sum_{i=1}^{k} \lambda_i^k c_i v_i, \tag{4.7}$$

the components associated with the dominant eigenvalue $\lambda_{\max}$ will grow more rapid than any other eigenvalue as $k \to \infty$, in practice making x_k more or less parallel with the dominant eigenvector $v_{\max}$. This means that the Krylov-sequence becomes an increasingly more ill-conditioned basis as k increases. To tackle the problem QR-factorisation is performed so that

$$Q_n R_n = K_n, \tag{4.8}$$

where Q_n is an orthonormal basis to $\mathcal{K}_n$. The similarity relation from Eq. (4.6) then reads

$$AQ_n R_n = Q_n R_n C_n. \tag{4.9}$$

By defining a new matrix $T \equiv R_n C_n R_n^{-1}$ we get the new important similarity equation

$$Q_n^T A Q_n = T. \tag{4.10}$$

As the similarity condition still holds, and the Hessenberg form remains under pre- or post-multiplication by another Hessenberg matrix, T is tridiagonal as well and gets the following appearance:

$$T = \begin{bmatrix} \alpha_1 & \beta_1 & 0 & \cdots & 0 \\ \beta_1^* & \alpha_2 & \beta_2 & & \vdots \\ 0 & \beta_2^* & \ddots & \ddots & 0 \\ \vdots & & \ddots & \alpha_{n-1} & \beta_{n-1} \\ 0 & \cdots & 0 & \beta_{n-1}^* & \alpha_n \end{bmatrix}. \tag{4.11}$$

The orthonormal basis Q_n from the QR-factorisation can be expressed as

$$Q_n = [q_1\ q_2 \dots\ q_n]. \tag{4.12}$$

Due to that A is similar to T, according to $AQ_n = Q_nT$, one column q_k will then satisfy the relation

$$Aq_k = \beta_{k-1}q_{k-1} + \alpha_k q_k + \beta_k^* q_{k+1}. \tag{4.13}$$

A re-ordering of Eq. (4.13) gives

$$q_{k+1} = \frac{1}{\beta_k^*}\left(Aq_k - \alpha_k q_k - \beta_{k-1}q_{k-1}\right), \tag{4.14}$$

and because q_{k+1} is normalized it can be concluded that

$$\beta_k^* = \left\|Aq_k - \alpha_k q_k - \beta_{k-1}q_{k-1}\right\|_2. \tag{4.15}$$

By multiplying q_k^T with Eq. (4.13) and the fact that $q_i^T q_j = \delta_{ij}$, the following relation is also found

$$\alpha_k = q_k^T A q_k. \tag{4.16}$$

These equations are all that is needed to successively build up both Q_n and T, leading to the Lanczos iteration in Algorithm 1. Furthermore: Since the vectors q_i of Q_n and the elements of T are acquired step by step, the Lanczos algorithm can thus be terminated at any iteration k.

Algorithm 1 The Lanczos algorithm

$q_0 = 0$
$\beta_0 = 0$
$x_0 =$ arbitrary nonzero starting vector
$q_1 = x_0 / \|x_0\|_2$
for $i = 1 \rightarrow k$ **do**
    $x_i = Aq_i$
    $\alpha_i = q_i^T x_i$
    $x_i = x_i - \beta_i q_{i-1} - \alpha_i q_i$
    $\beta_i = ||u_i||_2$
    $q_{i+1} = x_i/\beta_i$
end for

The known vectors $(q_i)_{i=1}^k$ thus form the columns of the matrix $Q_k = [q_1\ q_2\ ...\ q_k]$, fulfilling the relation

$$Q_n = \begin{bmatrix} Q_k & U_{n-k} \end{bmatrix}, \tag{4.17}$$

where U_{n-k} contains the $n-k$ remaining unknown vectors. Eq. (4.10) thus becomes

$$T = Q_n^T A Q_n = \begin{bmatrix} Q_k^T \\ U_{n-k}^T \end{bmatrix} A \begin{bmatrix} Q_k & U_{n-k} \end{bmatrix}$$
$$= \begin{bmatrix} Q_k^T A Q_k & Q_k^T A U_{n-k} \\ U_{n-k}^T A Q_k & U_{n-k}^T A U_{n-k} \end{bmatrix} = \begin{bmatrix} T_k & \widetilde{T}_k \\ \widetilde{T}_k^T & N_{n-k} \end{bmatrix}. \tag{4.18}$$

Because T is symmetric and tridiagonal the same must hold for T_k, and $\widetilde{T}$ can have at most one nonzero value, β_k, in its bottom left corner. Finally, N_{n-k} is a $(n-k)\times(n-k)$ tridiagonal matrix with unknown values. The eigenvalues of T_k are called *Ritz values* and the vectors $Q_k y$, where y is an eigenvector of T_k, are called *Ritz vectors*.

It can be shown that the Ritz values converge to the extreme eigenvalues of A, and the corresponding Ritz vectors converge to the corresponding eigenvectors, as k increases, which is the reason of the usefulness of the Lanczos algorithm.

Even for $k \ll n$ the Ritz pairs tend to be good approximations for the smallest and largest eigenpairs of A, saving a lot of computer power as the evaluation of T_k is far less intense than the one of A, if A is of considerable size. A more exhaustive discussion of the behaviour and convergence of Lanczos may be found in Appendix B.

4.2 Block Lanczos

The Lanczos method described in the last section may be extended to operate with blocks, i.e. small matrices, rather than single vectors, and is then commonly referred to as a *Block Lanczos method*. The motivation behind introducing a block generalization of the Lanczos algorithm is twofold: Firstly, when dealing with matrices of extremely large order, the computations are bound by memory access rather than the speed with which the arithmetic operations are performed. This suggests that it might be favourable to multiply the stored matrix with a small set of vectors rather than a single vector in the hopes of receiving a few number of matrix-vector multiplications essentially for free. Secondly, without modification the Lanczos algorithm is inherently unable to detect multiple eigenvalues. This a glaring issue especially in the context of quantum mechanics where quantum states are frequently degenerated, e.g. due to spin.

Recall that for a given symmetric matrix A and an arbitrary starting vector q, the basic Lanczos algorithm generates:

1. A sequence of vectors $(q_i)_{i=1}^s$ which constitutes an orthonormal basis for the associated Krylov subspace $\mathcal{K}_s(q,A) = \text{span}(q,Aq,\ldots,A^{s-1}q)$.

2. Sequences of scalars $(\alpha_i)_{i=1}^s$ and $(\beta_i)_{i=2}^s$ containing the diagonal and subdiagonal entries respectively, of a tridiagonal matrix T_s which is the projection of A onto to the Krylov subspace $\mathcal{K}_s(q,A)$.

Similarly, for a symmetric $n\times n$ matrix A and an arbitrary $n\times p$ orthonormal matrix Q the Block Lanczos method with block size p will generate:

1. A sequence of $n\times p$ orthonormal matrices $(Q_i)_{i=1}^s$ whose columns constitue an orthonormal basis for the *Block Krylov subspace* $\mathcal{K}_s(Q,A)$ spanned by the columns of the matrices $Q,AQ,\ldots,A^{s-1}Q$.

2. Sequences of $p\times p$ matrices $(A_i)_{i=1}^s$ and $(B_i)_{i=2}^s$ containing the diagonal and subdiagonal *blocks* respectively, of a $ps\times ps$ block tridiagonal matrix, which will also be denoted T_s for convenience. T_s is then the projection of A onto $\mathcal{K}_s(Q,A)$.

4.2.1 A Block Lanczos algorithm

In this section the basics properties of the algorithm are derived. The goal is to first construct an orthonormal basis for the Krylov subspace $\mathcal{K}_s(Q,A)$ and then proceed to show that projecting the matrix A onto this space yields a block tridiagonal matrix T_s. In some cases, rigorous proofs are omitted for simplicity. Curious readers are referred to [10] for a more detailed account of the Block Lanczos algorithm.

Suppose A is an $n\times n$ symmetric matrix, choose integers p, s such that $1 \leq ps \leq n$ and let Q be an arbitrary $n\times p$ matrix with orthonormal columns. Thus, p is the width of the initial block, commonly referred to as the *block size*, while s is the number of steps performed which determines the size of the basis. Set $Q_1 = Q$ and compute

$$Y_1 = AQ_1. \tag{4.19}$$

Next, let Z_2 be the projection of Y_1 onto the space orthogonal to the column space of Q_1, given by

$$Z_2 = (I - Q_1Q_1^T)AQ_1 = AQ_1 - Q_1Q_1^TAQ_1 = AQ_1 - Q_1A_1 \tag{4.20}$$

where $A_1 \equiv Q_1^TAQ_1$ is a $p\times p$ matrix. Whereas each column of Z_2 is guaranteed to be orthogonal to each of the columns in Y_1, the columns of Z_2 may not be mutually orthogonal. This can be remedied by a reduced QR-factorization

$$Z_2 = Q_2B_2, \tag{4.21}$$

yielding an $n \times p$ orthonormal matrix Q_2 and a $p \times p$ upper triangular matrix B_2. Note that $Q_2^T Q_1 = 0$ by construction since Q_2 is simply the normalized version of Z_2 which was the result of projeting AQ_1 onto the space orthogonal to Q_1.

The third Lanczos block Q_3 can now be obtained through the same sequence of steps. Compute $Y_2 = AQ_2$ and project the resulting matrix onto the space orthogonal to Q_2 and Q_1 yielding

$$Z_3 = (I - Q_1 Q_1^T - Q_2 Q_2^T) A Q_2 = A Q_2 - Q_2 A_2 - Q_1 Q_1^T A Q_2, \tag{4.22}$$

where $A_2 \equiv Q_2^T A Q_2$. Since $Q_2^T Q_1 = 0$ it follows from Eq. (4.21) that

$$B_2 = Q_2^T Z_2 = Q_2^T (A Q_1 - Q_1 A_1) = Q_2^T A Q_1. \tag{4.23}$$

Using the above equation and the fact that A is symmetric, Eq. (4.22) reduces to

$$Z_3 = A Q_2 - Q_2 A_2 - Q_1 B_2^T. \tag{4.24}$$

The third block Q_3 and the upper triangular matrix B_3 is then obtained by a QR-factorization, $Z_3 = Q_3 B_3$. For a general j, the correct generalization[2] of Eq. (4.24) accompanied by the usual QR-factorization is written

$$Q_{j+1} B_{j+1} = Z_{j+1} = A Q_j - Q_j A_j - Q_{j-1} B_j^T. \tag{4.25}$$

Where A_j is defined by

$$A_j \equiv Q_j^T A Q_j. \tag{4.26}$$

Thus, upon completion of j steps three sequences of matrices have been obtained, $(Q_i)_{i=1}^j$, $(A_i)_{i=1}^j$ and $(B_i)_{i=2}^j$, related by

$$\begin{aligned} Q_2 B_2 &= Z_2 = A Q_1 - A_1 Q_1 \\ Q_3 B_3 &= Z_3 = A Q_2 - A_2 Q_2 - Q_1 B_2^T \\ &\vdots \\ Q_{j+1} B_{j+1} &= Z_{j+1} = A Q_j - A_j Q_j - Q_{j-1} B_j^T \end{aligned} \tag{4.27}$$

Since the goal is to build an orthonormal basis for the Block Krylov space $\mathcal{K}_s(Q,A)$ and the columns of each individual Lanczos block are orthonormal by way of construction, it remains to show that distinct Lanczos blocks are mutually orthogonal. An inductive proof of this orthogonality can be found in Appendix C. The proof yields some useful side results which are stated here for future reference:

$$Q_{j-1}^T A Q_j = B_j^T, \tag{4.28}$$

[2]The approach here is to anticipate the result and show that this way of constructing new blocks yields the desired properties rather than attempting a straightforward derivation which can be a bit messy.

$$Q_j^T A Q_i = 0. \text{ for } i = 1, \ldots, j-2. \tag{4.29}$$

Thus, a total of s steps yields matrices Q_1,Q_2,...,Q_s such that their collected columns constitute an orthonormal set of vectors, which is in fact the sought-after basis for the Block Krylov space $\mathcal{K}_s(Q,A)$. This can be shown by a simple inductive argument but the proof is omitted for brevity. The columns of the sequence of Lanczos blocks $(Q_i)_{i=1}^s$ can be collected in a single matrix

$$\mathcal{Q}_s \equiv [Q_1\, Q_2 \ldots Q_s]\,. \tag{4.30}$$

Now, define a matrix T_s by

$$T_s \equiv \mathcal{Q}_s^T A \mathcal{Q}_s. \tag{4.31}$$

This can be interpreted as the matrix A being orthogonally projected onto the subspace spanned by $(Q_i)_{i=1}^s$, i.e. the Block Krylov space $\mathcal{K}_s(Q,A)$. This projection is represented by the matrix T_s[3] and it is natural to think of T_s as an $s\times s$ block matrix, the constituents of which are $p\times p$ blocks given by

$$\left(\mathcal{Q}_s^T A \mathcal{Q}_s\right)_{ij} = Q_i^T A Q_j = \begin{cases} A_i, & i = j \\ B_i, & i = j+1 \\ B_i^T, & i = j-1 \\ 0, & \text{otherwise} \end{cases} \tag{4.32}$$

which is just a restatement of Eqs. (4.26), (4.28) and (4.29) respectively. T_s may be written explicitly as

$$T_s = \begin{bmatrix} A_1 & B_2^T & 0 & \cdots & 0 \\ B_2 & A_2 & B_3^T & & \vdots \\ 0 & B_3 & \ddots & \ddots & 0 \\ \vdots & & \ddots & A_{s-1} & B_s^T \\ 0 & \cdots & 0 & B_s & A_s \end{bmatrix}, \tag{4.33}$$

which is the block equivalent of the tridiagonal matrix obtained in the original Lanczos algorithm. Following the conventions introduced in Section 4.1, an eigenvalue of T_s is called a Ritz value and if w is the corresponding eigenvector then $v \equiv \mathcal{Q}_s w$ is the Ritz vector. Furthermore, it is possible to prove [10] that these Ritz pairs converges to eigenpairs of A, same as for the vector Lanczos.

It is sometimes convenient to compactify the set of equations in Eq. (4.27) relating the three sequences of matrices generated by the Block Lanczos recurrence in Eq. (4.25) for an arbitrary j:

$$A\mathcal{Q}_j = \mathcal{Q}_j T_j + Q_{j+1} B_{j+1} E_j^T, \tag{4.34}$$

[3] or, to put it in other words: computing T_s amounts to expressing A in the new basis derived for the Krylov subspace. It is a remarkable fact that even if the dimension of this space is comparatively small, eigenvalues of T_s still provides good approximations of the most extreme eigenvalues of A.

where E_j^T is the $n \times p$ matrix whose $n - p$ first rows are all zero and the last $p \times p$ block is I_p. This is sometimes referred to as a *Block Lanczos decomposition* and the expression may be verified by direct substitution using the defining equation for $\mathcal{Q}_j$ and E_j^T.

The results obtained thus far may be condensed into a Block Lanczos algorithm. To reiterate, suppose A is an $n \times n$ symmetric matrix, integers p, s have been chosen such that $p \geq 1$ and $1 \leq ps \leq n$ and Q is an arbitrary orthonormal $n \times p$ matrix Q. Compute matrices $\mathcal{Q}_s$ and T_s through the steps of Algorithm 2. Note that the order of the computation in this algorithm differs somewhat from the original construction. From a theoretical viewpoint, this rearrangement makes no difference but is employed rather because it is less sensitive to rounding errors [11].

Algorithm 2 The Block Lanczos algorithm

1: $Q_0 = 0$
2: $B_1 = 0$
3: $Q_1 = Q$
4: **for** $i = 1,2,\ldots,s$ **do**
5: $\quad U_i = AQ_i - Q_{i-1}B_i^T$
6: $\quad A_i = Q_i^T U_i$
7: $\quad Z_{i+1} = U_i - Q_i A_i$
8: $\quad$ Compute the reduced QR-factorization $Q_{i+1}B_{i+1} = Z_{i+1}$
9: **end for**

4.3 Restarted Lanczos methods

In practical applications, the amount of memory available for a given computation is limited. Depending on the dimension of the problem, the Lanczos method utilized, the number of eigenpairs sought and so forth, it may prove inconvenient or impossible to build a large enough Krylov space to attain the desired level of convergence. Indeed, as the dimension of the Krylov space increases, so does the cost of each increment (due to the need for reorthogonalization measures, see Section 4.4) and for this reason alone it may be desirable to limit the allowed dimension of the Krylov space.

Difficulties of this nature are commonly circumvented by restarting the Lanczos procedure and is then referred to as a *restarted Lanczos method* or an *iterative Lanczos method*. A restarted Lanczos method can be loosely defined as a Lanczos procedure where, after a specified number of steps, the algorithm is halted and restarted using information from the previous run.

Algorithm 3 Restarted Block Lanczos algorithm

1: Generate an arbitrary orthonormal starting block Q_1 of width at least r.
2: Compute matrices T_s and $\mathcal{Q}_s$ using Algorithm 2.
3: Compute the r least eigenvalues μ_i and corresponding eigenvectors w_i of T_s.
4: Compute $v_i = \mathcal{Q}_s w_i$, $i = 1, \ldots, r$.
5: Estimate the accuracy of the Ritz pairs (μ_i, v_i) as approximations to eigenpairs of A.
6: **if** Convergence has occurred **then**
7: **stop**
8: **else**
9: Set $Q = [v_1 \ldots v_r]$ and repeat from step 2.
10: **end if**

Consider as an example the Block Lanczos algorithm described in the last section, Algorithm 2. Suppose the algorithm is initialized with a random starting block and terminated once the dimension of the Krylov space has reached one maximum tolerant dimension. The resulting block tridiagonal matrix T and the generated Lanczos blocks are then used to compute Ritz pairs. If the desired level of convergence has not been attained, the Ritz vectors corresponding to the desired eigenvalues are used as the starting block for the next Lanczos run. This can be summarized in Algorithm 3 for computing the r least (or largest) eigenvalues of a matrix A.

Note that algorithm Algorithm 3 hinges on using a block size which is greater than or equal to desired number of eigenvalues.

This restarting scheme can be expanded on somewhat: Suppose that after a Block Lanczos run, only a subset the Ritz vectors $v_1, \ldots, v_k$ where $k < r$ have converged. Since Algorithm 3 makes no distinction between these vectors and the remaining $v_{k+1}, \ldots, v_r$, converged Ritz pairs will be recomputed during subsequent runs.

It is possible to rid Algorithm 3 of this superfluous computational expense using a scheme where Ritz vectors that have converged sufficiently during a particular run are effectively banished during the following runs. The easiest way to accomplish this is to compute an eigendecomposition of T_s at the end of each Lanczos run and store the converged Ritz vectors while using those not yet converged as starting approximations for the next run. This results in Algorithm 4 for this modification of the restarted Block Lanczos method.

Thus, when a few Ritz vectors have converged in Algorithm 4, these vectors will be ignored in the formation of the new starting block Q_1 and this new

Algorithm 4 Restarted Block Lanczos algorithm (modified)

1: Generate an arbitrary orthonormal starting block Q_1 of width at least r.
2: $m = 0$
3: **while** $m < r$ **do**
4: Compute matrices T_s and $\mathcal{Q}_s$ using Algorithm 2.
5: Compute the r least eigenvalues μ_i and corresponding eigenvectors w_i of T_s.
6: Compute $v_i = \mathcal{Q}_s w_i$, $i = 1, \ldots, r$.
7: Estimate the accuracy of the Ritz pairs (μ_i, v_i) as approximations to eigenpairs of A.
8: Suppose that k Ritz vectors have converged, store these vectors and increase $m = m + k$
9: $Q_1 = [v_{m+1} v_{m+2} \ldots v_r]$
10: **end while**

block will in fact be orthogonal to the set of converged Ritz vectors since any eigenvectors computed from T_s are orthogonal and will remain so under multiplication with the unitary matrix $\mathcal{Q}_s$. In ideal settings, this orthgonality would be preserved during the formation of new Lanczos during step 4 in Algorithm 4 where Algorithm 2 is used to compute new the T_s and $\mathcal{Q}_s$. In practice however, the orthogonality is quickly lost due to round-off errors which introduce components in the directions the banished Ritz vectors. This general phenomenon is more thoroughly explained in Section 4.4 and an easy albeit not so elegant way to purge the Lanczos blocks from these unwanted components is to orthgonalize each newly generated block against the set of converged Ritz vectors.

In the case of the single vector Lanczos a different approach is necessary. For instance, after a particular run a linear combination of the current Ritz vectors can used to restart the algorithm. This is just a special case of applying a *filter polynomial* to the starting vector. To see how this works, suppose that q is an arbitrary vector and let $(\lambda_i, v_i)_{i=1}^n$ be a complete set of eigenpairs of A. q can now be expanded in terms of the eigenvectors v_i:

$$q = \sum_{i=1}^{n} c_i v_i \ . \tag{4.35}$$

Next, let $P(x)$ be an arbitrary polynomial and denote by $P(A)$ the corresponding matrix polynomial which is the result of substituting x for A everywhere in P. Multiplying q by this polynomial and using Eq. (4.35)

yields

$$P(A)q = P(A)\sum_{i=1}^{n} c_i v_i = \sum_{i=1}^{n} c_i P(A)v_i = \sum_{i=1}^{n} c_i P(\lambda_i)v_i \,. \tag{4.36}$$

Now, suppose that only the k first of the eigenpairs of A are to be calculated and split up the sum in Eq. (4.36) into two parts:

$$P(A)q = \sum_{i=1}^{k} c_i P(\lambda_i)v_i + \sum_{i=k+1}^{n} c_i P(\lambda_i)v_i \,. \tag{4.37}$$

Thus, by using $P(A)q$ as the starting vector for the next Lanczos run and choosing an appropriate polynomial P, it is possible to minimize the components of $P(A)q$ along the directions of the unwanted eigenvectors $v_{k+1},\ldots,v_n$. In this case, P is referred to as a filter polynomial.

Example. If $\mu_{k+1},\ldots,\mu_n$ are Ritz values corresponding to the unwanted end of the spectrum, an obvious choice of P is given by

$$P(x) = (x-\mu_{k+1})\cdots(x-\mu_n). \tag{4.38}$$

Inserting this into equation Eq. (4.36) yields

$$P(A)q = \sum_{i=1}^{n} c_i \prod_{j=k+1}^{n} (\lambda_i - \mu_j)v_i \,. \tag{4.39}$$

Since the Ritz values $\mu_i \approx \lambda_i$, terms in Eq. (4.39) containing the factor $(\lambda_i - \mu_i)$ are small. But these factors can only appear for $i \geq k+1$ thus damping only the unwanted components v_{k+1}. ■

Computing a filter polynomial is usually a formidable task. However, without going into further details, if q is used to start the Lanczos algorithm it is possible to compute the Lanczos decomposition of $P(A)q$ using a QR-method with shifts without introducing any further matrix products involving A. This leads to a scheme for restarting the Lanczos algorithm known as *implicit restarting* and readers are referred to [12] for a detailed account. It is possible to extend the technique described above to include the Block version of the Lanczos algorithm as well, see [13].

A quick note on terminology: In literature, restarted Lanczos methods that that uses some kind of filter polynomial without the addition of the shifted QR steps, are sometimes (rather confusingly) referred to as *explicitly restarted*. Similarly, Block methods such as the one described in Algorithm 3 are occasionally called explicitly restarted as well.

4.4 Loss of orthogonality

Up until this point, the single vector and the Block Lanczos methods have been discussed in the context of infinite precision arithmetic. This section will discuss the issues that arise in finite-precision arithmetic, i.e. when the actual computations are made using computers.

In the following discussion the single vector Lanczos method will be regarded as a special case of the Block Lanczos method with the block size set to one unless stated otherwise. Recall that new orthonormal Lanczos blocks are constructed via a three term recurrence, see Eq. (4.25) and that these blocks were shown to be mutually orthogonal. In an ideal setting, this means that if the most recently generated block is Q_{j+1} then for all $k \leq j$

$$\left|\left|Q_k^T Q_{j+1}\right|\right| = 0, \tag{4.40}$$

where $||\ldots||$ denotes the spectral norm or 2-norm of a matrix A defined by

$$||A|| \equiv \max\left\{\frac{||Ax||}{||x||} : x \in \mathbb{R}^n\right\}. \tag{4.41}$$

In practice however, round-off errors incurred in computing the matrix products will result in what is called a (global) *loss of orthogonality* among the Lanczos blocks. This can be clarified by a simple geometric argument: Suppose that when a Lanczos block Q_{j+1} is computed, a round-off error introduces small components in the directions of some vectors contained in some of its predecessors Q_j,$Q_{j-1}, \ldots$, which are assumed to be perfectly orthogonal, so that Q_{j+1} is no longer perfectly orthogonal to these blocks. Next Q_{j+2} will be computed but according to Eq. (4.25) only components in the directions contained in Q_{j+1} and Q_j will be purged and thus Q_{j+2} will not be orthogonal to $Q_{j-1}, \ldots$. Thus, round-off errors propagate as new blocks are added. Furthermore, it can be argued that these errors not only propagate but may be amplified during the formation process [14].

The loss of orthogonality occurs surprisingly fast as the example below will illustrate. The level of orthogonality between distinct blocks is measured by $\left|\left|Q_k^T Q_{j+1}\right|\right|$ in Eq. (4.40) and it will be shown that this quantity will quickly exceed the unit round-off ϵ_{mach}[4] as new blocks are added.

Example. *(Loss of orthogonality)* Consider a 119240×119240 symmetric matrix representing the Hamiltonian of a quantum mechanical system consisting of 10 interacting bosons. The block size is set to $p = 4$ and the block Lanczos algorithm has been run for $j = 20$ steps. The orthogonality of the most recently generated block against its predecessors is

[4] Also referred to as machine-epsilon and is roughly speaking a bound on the relative error in the used floating point representation of a real number.

measured by $\log_{10}\left(\left|\left|Q_k^T Q_j\right|\right|/\epsilon_{\text{mach}}\right)$ and displayed for each step in Table 4.1. For instance, the third row contains $\log_{10}\left(\left|\left|Q_1^T Q_3\right|\right|/\epsilon_{\text{mach}}\right)$ and $\log_{10}\left(\left|\left|Q_2^T Q_3\right|\right|/\epsilon_{\text{mach}}\right)$. This measure indicates by how many orders of magnitude the unit round-off $\epsilon_{\text{mach}} = 2.2204 \times 10^{-16}$ is exceeded. ■

The formulas previously derived for the Block Lanczos recursion, see Section 4.2.1, remain valid for small errors up to about the size of the unit round-off. Thus, referring to the above example, it is clear that losses of orthogonality introduces errors which are not negligible since they surpass the unit round-off by several orders of magnitude. Left unattended, the loss of orthogonality among Lanczos blocks results in the appearance of multiple copies of one or more eigenvalues, even if the true multiplicity of the eigenvalue is one. This is sometimes referred to as the *ghost eigenvalue problem*. Furthermore, given enough iterations the basis for the Krylov space eventually deteriorates completely and the resulting spectrum of T may be very different from that of A.

Table 4.1. *A table illustrating the global loss of orthogonality among Lanczos blocks. The* (j,k)-th *entry is given by* $\log_{10}\left(\left|\left|Q_k^T Q_j\right|\right|/\epsilon_{\text{mach}}\right)$ *and shows by how many orders of magnitude the unit round-off is exceeded.*

Lanczos step j																		
1	0																	
2	3																	
3	2	3																
4	3	2	3															
5	3	3	3	3														
6	3	3	3	3	3													
7	3	3	3	3	3	3												
8	3	3	3	3	3	3	3											
9	4	4	4	4	3	3	3	3										
10	4	4	4	4	4	4	3	3	3									
11	5	5	5	4	4	4	4	3	3	3								
12	5	5	5	5	5	5	4	4	3	3	3							
13	6	6	6	6	5	5	5	4	4	3	3	3						
14	7	7	7	6	6	6	5	5	5	4	4	3	3					
15	7	7	7	7	7	7	6	6	5	5	5	4	3	3				
16	8	8	8	8	8	7	7	7	7	6	6	5	5	4	3			
17	9	9	9	9	8	8	8	7	7	7	6	6	5	5	4	4		
18	9	9	9	9	9	9	8	8	8	7	7	7	6	5	5	4	3	
19	10	10	10	10	10	9	9	9	8	8	8	7	7	6	5	4	4	3
20	11	11	11	10	10	10	10	9	9	8	8	8	7	6	6	5	4	3	2

To adress both these problems, it is almost always necessary to include some degree of reorthogonalization of the generated Lanczos blocks and the options available are discussed below. With the exception of Section 4.4.1, the schemes outlined in Sections 4.4.2, 4.4.3, 4.4.4 and 4.4.5 are all compatible with the restarted Lanczos-methods described in Section 4.3.

4.4.1 No reorthogonalization

It has been argued that the appearance of multiple copies of eigenvalues is indicative of the convergence of the associated Ritz vectors. This has led some authors [15], to propose a single vector Lanczos using no reorthogonalization. Spurious as well as ghost eigenvalues can be dealt with but this generally requires additional computation. For Block Lanczos-based methods this is generally not viable since the loss of orthogonality is more pronounced and occurs rapidly.

4.4.2 Full reorthogonalization

An obvious way to maintain orthogonality among the Lanczos blocks is to reorthogonalize each newly generated block against all the previously computed blocks. This is accomplished using the Gram-Schmidt process or some improvement thereof, such as the modified Gram-Schmidt (MGS). As an example, consider the basic Block Lanczos algorithm presented in Section 4.2.1 with full reorthogonalization (FRO) using MGS,

Algorithm 5 Block Lanczos with full reorthogonalization

1: Perform steps 1 through 13 of Algorithm 2.
2: **for** $j = 1, \ldots, i$ **do**
3: Orthogonalize Q_{i+1} against Q_j using MGS.
4: **end for**

While this option solves the issue of maintaining the orthogonality in a satisfactory manner, it comes with two major drawbacks. Firstly, a full reorthogonalization against all previously generated Lanczos blocks each iteration significantly increases the amount of arithmetic operations required, and this additional cost increases with the number of steps since there will be more Lanczos blocks available. Secondly, it requires, at each step of reorthogonalization, the presence of all previously generated blocks in main memory. This is a crucial point since much of the appeal of Lanczos-based methods lies in their relative cheapness when considering the amount of high-speed storage space required.

4.4.3 Partial reorthogonalization

A third option is to maintain just enough orthogonality among the Lanczos blocks in order to avoid the problems associated with no reorthogonalization while reducing the amount of arithmetic operations entailed by the FRO. While the details of the scheme used for maintaining near-orthogonality among the Lanczos blocks varies between different implementations they usually share the same basic features. One such scheme is outlined below for the Block Lanczos algorithm[5].

The main idea is to simulate the loss of orthogonality among the blocks and only perform a reorthogonalization when the estimated losses exceed some level of preset tolerance. The first goal is to get a theoretical bound on the deviation of the product $Q_k^T Q_{j+1}$ from the zero matrix for $k < j-1$. Consider the main recurrence Eq. (4.25) in Algorithm 2 where an additional matrix F_j has been introduced to account for round-off errors[6],

$$Q_{j+1}B_{j+1} = AQ_j - Q_jA_j - Q_{j-1}B_j^T + F_j. \tag{4.42}$$

Premultiplying both sides of this expression with Q_k^T where $k < j-1$ yields

$$Q_k^T Q_{j+1}B_{j+1} = Q_k^T AQ_j - Q_k^T Q_j A_j - Q_k^T Q_{j-1}B_j^T + Q_k^T F_j. \tag{4.43}$$

Define $W_{jk} \equiv Q_k^T Q_j$ and rewrite the above equation in terms of W_{jk},

$$W_{j+1k}B_{j+1} = Q_k^T AQ_j - W_{jk}A_j - W_{j-1k}B_j^T + Q_k^T F_j. \tag{4.44}$$

A useful bound on W_{jk} should contain only quantities that are either already accessible or easily computable and thus the first term on the right hand side of Eq. (4.44) containing the matrix A must be eliminated. This is done by permuting the indices $(j,k) \to (k,j)$ and rearranging the terms in Eq. (4.44):

$$Q_j^T AQ_k = W_{k+1j}B_{k+1} + W_{kj}A_k + W_{k-1j}B_k^T - Q_j^T F_k. \tag{4.45}$$

Taking the transpose of this expression and using the fact that A is symmetric yields

$$Q_k^T AQ_j = B_{k+1}^T W_{jk+1} + A_k W_{jk} + B_k W_{jk-1} - F_k^T Q_j. \tag{4.46}$$

Substitute Eq. (4.46) for the first term on the right hand side of Eq. (4.44):

$$W_{j+1k}B_{j+1} = B_{k+1}^T W_{jk+1} + A_k W_{jk} + B_k W_{jk-1} - W_{jk}A_j - W_{j-1k}B_j^T + G_{jk} \tag{4.47}$$

[5]The single vector Lanczos is retained as the special case when the block size $p = 1$.

[6]Thus $Q_{j+1}, A_j, \ldots$ now refer to the actual values computed rather than the ideal quantities of the previous sections.

where the term $G_{jk} \equiv Q_k^T F_j - F_k^T Q_j$ accounts for the round-off errors. Eq. (4.47) may now be turned into an upper bound on W_{jk} by taking the spectral norm[7]:

$$\begin{aligned} ||W_{j+1k}|| \leq \left|\left|B_{j+1}^{-1}\right|\right| \Big[&||B_{k+1}||\,||W_{jk+1}|| + ||B_k||\,||W_{jk-1}|| \\ &+ ||B_j||\,||W_{j-1k}|| + (||A_j|| + ||A_k||)\,||W_{jk}|| + ||G_{jk}||\Big]. \end{aligned} \tag{4.48}$$

Here, repeated use has been made of two elementary inequalities satisfied by the spectral norm for any two matrices A and B.

1. *The triangle inquality*: $||A + B|| \leq ||A|| + ||B||$
2. $||AB|| \leq ||A||\,||B||$

For notational convenience, define $\alpha_k \equiv ||A_k||$, $\beta_k \equiv ||B_k||$, $\tilde{\beta_k} \equiv 1/\sigma_{\min}(B_k)$, where $\sigma_{\min}(B_k)$ is the smallest singular value of B_k, and let ω_{jk} denote the bound on $||W_{jk}||$. Note that in the special case where the block size $p = 1$ the quantities α_k and β_k are the same ones encountered in Section 4.1 and no evaluation of matrix norms is required. The bound in Eq. (4.48) can now be used in Algorithm 6 for simulating the loss of orthogonality between generated Lanczos blocks, where s denotes the step length, n is the dimension of the matrix A, p is the block size and ϵ_{mach} is the unit round-off.

Algorithm 6 Estimating the loss of orthogonality

At each Lanczos step $i > 2$ after computing Q_{j+1} **do**
  $\epsilon_s \equiv \epsilon_{\text{mach}} p\sqrt{n}$
  $\omega_{11} = \epsilon_s$
  $\omega_{21} = \epsilon_s \tilde{\beta_2}$
  $\omega_{22} = \epsilon_s$
  for $j = 2,3,\ldots,s$ **do**
    $\omega_{j+1j+1} = \epsilon_s$
    $\omega_{j+1j} = \epsilon_s$
    for $k = 1,\ldots,j-1$ **do**
      $\omega_{j+1k} = \tilde{\beta_{j+1}}\big[\beta_{k+1}\omega_{jk+1} + \beta_k\omega_{jk-1} + \beta_j\omega_{j-1k} + (\alpha_j + \alpha_k)\,\omega_{jk}\big]$
    end for
  end for

Here, the quantity ϵ_s is a bound on the rounding error, i.e. $||G_{jk}|| < \epsilon_s$ [14]. At any given step $j > 2$ of the Block Lanczos algorithm bounds on the quantity $||Q_{j+1}Q_k||$, $k = 1,\ldots,j-1$, are computed using the above algorithm

[7]Except for the vector Lanczos, i.e. when $p = 1$. In this case Eq. (4.47) is already scalar and can be used without further modification.

and a natural choice is to reorthogonalize the two most recent Lanczos blocks against all previously generated blocks only when the maximum of these bounds exceeds a preset tolerance. Numerical experiments indicate that a suitable choice of this tolerance is $\sqrt{\epsilon_{\text{mach}}}$ [16], i.e. a full reorthogonalization step is performed whenever

$$\max_x \omega_{j+1k} \geq \sqrt{\epsilon_{\text{mach}}} \tag{4.49}$$

Putting all this together gives Algorithm 7 for Block Lanczos with partial reorthogonalization.

Algorithm 7 Block Lanczos with partial reorthogonalization

1:
2: At each Lanczos step $i > 2$ after computing Q_{j+1} **do**
3: Update the ω recurrence according to Algorithm 6.
4: $\omega_{\max} = \max_x \omega_{j+1k}$
5: **if** $\omega_{\max} \geq \sqrt{\epsilon_{\text{mach}}}$ **then**
6: **for** $k = 1, \ldots, j-1$ **do**
7: Orthgonalize Q_j against Q_k using MGS.
8: Orthgonalize Q_{j+1} against Q_k using MGS.
9: **end for**
10: Orthgonalize Q_{j+1} against Q_j
11: $\omega_{j+1k} = \epsilon_s$, $k = 1, \ldots, j$ ▷ Update the ω-recurrence
12: $\omega_{jk} = \epsilon_s$, $k = 1, \ldots, j$
13: **end if**

Note that step 9 in Algorithm 7 introduces a *local reorthogonalization* (LRO) between the blocks Q_{j+1} and Q_j after each step of PRO. In addition, an extra LRO is done during each ordinary Block Lanczos step as well. This is due to previous results indicating a strong correspondence between the global loss of orthogonality among Lanczos blocks and the local loss of orthogonality between adjacent blocks, see [16]. This way, LRO is guaranteed to hold which justifies the choice of $\omega_{j+1j} = \epsilon_s$ in Algorithm 6. The level of orthogonality maintained between the blocks using this scheme (i.e. $\sqrt{\epsilon_{\text{mach}}}$) is sometimes referred to as *semi-orthogonality*. This particular choice of also has a theoretical motivation and it can be proved that under this level of orthogonality, the (block) tridiagonal matrix T is up to round-off the orthogonal projection of A onto the Krylov space, which is the real reason behind the success of PRO [14].

To summarize, this PRO scheme simulates the loss of orthogonality among Lanczos blocks and orthogonalizes the two most recently generated blocks against all previously computed blocks, whenever the condition in Eq. (4.49)

is violated. On top of this a LRO is performed during every step to maintain a working-precision orthogonality between Q_{j+1} and Q_j. At each step j, the Algorithm 6 requires the computation of the spectral norms of two $p\times p$ matrices and one $pj\times pj$ matrix, which is cheap compared to an orthogonalization using Gram-Schmidt when the dimension of the matrix is large. While the PRO lowers the arithmetic operations count considerably, all generated Lanczos blocks still have to be kept in main memory (or possibly in secondary storage).

4.4.4 Modified partial reorthogonalization

In the interest of minimizing the total number of orthogonalizations performed, one question concerning the PRO-scheme has yet to be addressed. Algorithm 7 above certainly reduces the number of orthogonalization steps[8] but once the condition of semi-orthogonality is violated the following reorthogonalization is done against all previously computed Lanczos blocks which might very well involve unnecessary calculations. If this is indeed the case, the next question is how to determine an appropriate subset of Lanczos blocks to use for orthogonalization. The ω-simulation described in Algorithm 6 provides a natural starting point for such investigations since it contains all the information of the global loss of orthogonality. Indeed, a few observations can be made immediately from the main recurrence governing the bound $\omega_{j+1,k}$:

$$\omega_{j+1k} = \tilde{\beta}_{j+1}\left(\beta_{k+1}\omega_{jk+1} + \beta_k\omega_{jk-1} + \beta_j\omega_{j-1k} + (\alpha_j + \alpha_k)\,\omega_{jk}\right). \quad (4.50)$$

From the right hand side of this equation it is plain that terms contributing to ω_{j+1k} are not limited to ω_{jk} and ω_{j-1k} but will also include the neighboring terms ω_{jk-1} and ω_{jk-1}. This suggests that for a fixed k, ω_{j+1k} will remain reasonably small for the next few iterations only if ω_{jk-1} and ω_{jk+1} are made small as well. However, these terms are themselves dependent on neighboring terms (and so forth). The easiest way to go about deciding the actual number of neighboring terms to include is to devise an algorithm which simply selects a bunch of these terms, the size of which is determined by a free parameter η. More specifically, at each iteration j of the Block Lanczos algorithm, the ω-recurrence is updated according to Algorithm 6 and rather than reorthogonalizing whenever Eq. (4.49) is violated, a search is done to reveal the individual offending ω_{j+1k} for $k = 1,\ldots,j-1$. Along with each of these offending k, an interval containing some of its neighbors is determined and all corresponding blocks falling within one ofthese intervals are then selected for orthogonalization. For a fixed k, the size of the

[8] Numerical evidence for this claim will be presented in Chapter 5.

corresponding interval $[l_k,u_k]$ is determined by the bounds l_k and u_k defined by

$$l_k = \min_i \omega_{j+1i} \geq \eta \tag{4.51}$$

and

$$u_k = \max_i \omega_{j+1i} \geq \eta. \tag{4.52}$$

This leads to a slightly altered version of the PRO-scheme, sometimes referred to as modified partial reorthogonalization (MPRO) and is implemented as follows:

Algorithm 8 Block Lanczos with modified partial reorthogonalization

1: Perform steps 1 through 9 of Algorithm 2.
2: Update the ω recurrence according to Algorithm 6.
3: **while** $k < j - 1$ **do**
4: **if** $\omega_{j+1k} \geq \sqrt{\epsilon_{\text{mach}}}$ **then**
5: Determine the interval $[l_k,u_k]$ defined by Eqs. (4.51) and (4.52).
6: $k = k + u_k$ ▷ Avoid overlapping intervals
7: **else**
8: $k = k + 1$
9: **end if**
10: **end while**
11: **for** each interval $[l_k,u_k]$ **do**
12: Orthogonalize Q_{j+1} against Q_l, for all $l \in [l_k,u_k]$
13: $\omega_{j+1l} = \epsilon_s$
14: $[l_k,u_k] = [l_k - 1,u_k + 1]$
15: **end for**

Furthermore, whenever a reorthogonalization step occurs during an iteration in Algorithm 8 it is immediately repeated during the next iteration to guarantee that semi-orthogonality holds between the most recently generated block and all its ancestors. This was not explicitly stated in Algorithm 8 so as to avoid unnecessary complications.

It is important to note that by no means is it clear that this modified scheme is actually an improvement of the original scheme as described in Algorithm 7. Rather, an attempt can be made to minimize the number of orthogonalizations for a given problems by choosing different values of the free parameter η in the interval $[\epsilon_s,\sqrt{\epsilon_{\text{mach}}}]$. Whether this results in any significant decrease in orthogonalizations compared to the original PRO-scheme can only be determined by numerical experiments.

4.4.5 Selective reorthogonalization

Another scheme for reducing the amount of orthogonalizations is *selective reorthogonalization* where new Lanczos vectors are orthogonalized against a small set of Ritz vectors that have very nearly converged. This technique has been extensively treated by B. Parlett and D. Scott, see [17]. Available to both the single vector as well as the Block Lanczos algorithm, selective reorthogonalization is based on the following theorem originally proved by Paige. The version stated here is that of G. Golub and C. Van Loan [18].

Theorem 4.4.1 (*Paige, 1971) Suppose that a tridiagonal matrix T_j and an orthonormal basis $Q_j = [q_1, \ldots, q_j]$ have generated by the Lanczos algorithm. Let $\mu_1, \ldots, \mu_j$ denote the Ritz values computed from T_j and let $V = [v_1, \ldots, v_j]$ be a matrix formed from the corresponding Ritz vectors, $v_k = \mathcal{Q}_j w_k$, $k = 1, \ldots, j$ where w_k is an eigenvector of T_j. Then for $k = 1, \ldots, j$,*

$$\left|q_{j+1}^T v_k\right| \approx \frac{\epsilon_{\text{mach}} \, ||A||}{|\beta_k| \, |v_{jk}|}$$

A direct consequence of this theorem is that the most recently generated Lanczos vector q_{j+1} has a component in the direction of any computed Ritz vector assuming it has converged sufficiently. The right hand side of the above equation also indicates that the size of this component is in general not negligible. Indeed, more thorough investigations reveal that the losses of orthogonality occurring in the directions of converged Ritz vectors tend to dominate other directions, suggesting that instead of a FO or PRO, a reorthogonalization against a small subset of converged Ritz vectors might prove sufficient. This is the main idea behind selective reorthogonalization (SO), for the details of its realisation the curious reader is referred to [17].

4.5 Categorizing different Lanczos methods

The different aspects and variations of the Lanczos algorithms discussed in the previous sections, both theoretical and practical, may be summarized in the following table for specifying a particular Lanczos-based method. In the column specifying the different reorthgonalization techniques, the PRO and MPRO are grouped together under the same label "Limited: Partial".

Type of choice	**Choices available**
Type of recursion	-Vector -Block
Restart	-None -Implicit restart (Vector) -Explitict restart (Block) -Implicit restart (Block)
Reorthogonalization	-None -Full -Limited: Partial -Limited: Selective -Limited: Local

5 Investigations of Lanczos and Block Lanczos Methods

In Chapter 4 it was found that large symmetric sparse eigenproblems (LSSEs) are more efficiently dealt with using a Lanczos-based method. However as was also noted, there exists a wide range of such methods that share a common theoretical foundation but ultimately differ in some respect or another. Despite years of intense research there is not, and arguably will never be, a black box Lanczos method where the user simply inputs a desired number of eigenvalues to be computed regardless of the problem at hand. A physicist or engineer looking to solve a LSSE is thus faced with the difficult choice of which particular brand of Lanczos-solver to use, a decision that requires him or her to factor in the specific details of the problem. For the purpose of solving the LSSEs that arise in the context of quantum mechanical few- and many-body problems, as studied in this work, there are several important factors that directly limit or affect this choice:

1. Typically, no more than 10-20 of the algebraically smallest eigenvalues have to be computed.
2. Eigenstates should be available for computation without too much effort.
3. Multiple eigenvalues should be computed naturally since energy-levels are frequently near-degenerate.
4. The dimension of the matrix representing the Hamiltonian may be of orders larger than 10^9, severely limiting the amount of information that can be kept in main memory or stored on disk.

The remainder of this chapter is devoted to a numerical study of the Lanczos and Block Lanczos Methods, including different techniques for restoring orthogonality among Lanczos vectors, convergence analysis, model spaces and restarting schemes. The aim is then to use the results from this investigation in conjunction with the knowledge of the quantum mechanical problem in order to establish a set of recommendations for a future implementation of an eigensolver, see Chapter 9. Throughout this chapter, all numerical tests are based on MATLAB implementations of the basic algorithms described in Chapter 4. Furthermore, all test matrices were generated by the NSCMb code unless stated otherwise.

5.1 Stopping criterion and convergence properties

Before turning to any numerical tests concerning block size, reorthgonalization schemes and so forth, it is instructive to take a quick look at the typical behavior of the convergence of some of the algebraically smallest eigenvalues. Consider the basic vector Lanczos described in Section 4.1, Fig. 5.1 shows the convergence of the four algebraically smallest eigenvalues of a 5147×5147 matrix A, measured by the absolute error $\epsilon_{\text{abs}} = |\lambda - \tilde{\lambda}|$[1] as a function of the number of steps. Accurate eigenvalues λ_i, $i = 1, 2, 3, 4$ were computed to working-precision using a conventional QR-based eigensolver.

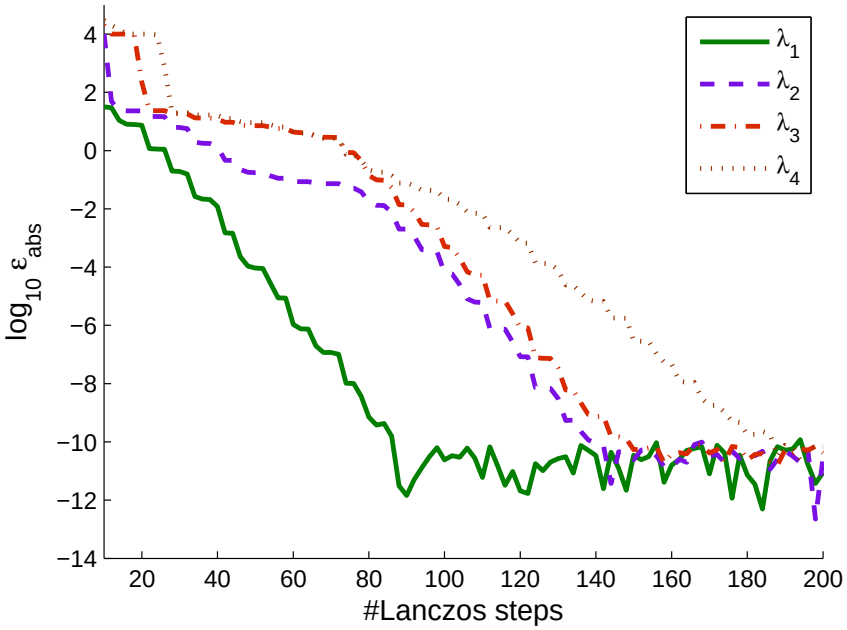


Figure 5.1. *The convergence of the four algebraically smallest computed eigenvalues $\tilde{\lambda}_i$, i = 1,2,3,4 towards the true eigenvalues λ_i, measured by the absolute error $\epsilon_{\text{abs}} = |\lambda_i - \tilde{\lambda}_i|$ (shown on a logarithmic scale) as a function of the number of Lanczos steps.*

Note that the rate of the convergence of the eigenvalues in Fig. 5.1 , i.e. the slope of the lines, occasionally reaches a plateau where for the next few Lanczossteps almost no change is observed in the absolute error. It is also clear that the smaller the eigenvalue, the faster the convergence. This behavior is characteristic of all Lanczos methods and a quantitative explanation is provided in Appendix B.

[1] This is not to be confused with ϵ_{mach} which is used to denote the unit round-off

5.1.1 A stopping criterion

In order for a practical Lanczos method to be efficient a stopping criterion is required. A natural choice is to terminate a particular Lanczos run whenever

$$\left|\left|A\tilde{v} - \tilde{\lambda}\tilde{v}\right|\right| \leq \text{tol} \tag{5.1}$$

for an approximate eigenpair ($\tilde{\lambda}$,$\tilde{v}$) and some tolerance tol specified by the user. The quantity on the left hand side of Eq. (5.1) is sometimes referred to as the *residual norm*. This seemingly involves an unwanted computation of the matrix-vector product Av which can in fact be circumvented using information generated by the Lanczos run. In the following derivation, any tildes denoting an approximate eigenvalue will be dropped for notational convenience.

Suppose that the algorithm is paused after j steps yielding a Block Lanczos decomposition (see Chapter 4, Section 4.2.1)

$$A\mathcal{Q}_j = \mathcal{Q}_j T_j + Q_{j+1} B_{j+1} E_j^T. \tag{5.2}$$

Let μ be an eigenvalue of T_j and w the corresponding eigenvector. Recall that this eigenpair is related to an approximate eigenpair (λ,v) of A by $\lambda = \mu$ and $v = \mathcal{Q}_j w$. The quantity $||Av - \lambda v||$ can now be expressed using these two identities and the decomposition in Eq. (5.2):

$$\begin{aligned} ||Av - \lambda v|| &= ||A\mathcal{Q}_j w - \mathcal{Q}_j w\lambda|| \\ &= \left|\left|\mathcal{Q}_j T_j w + Q_{j+1} B_{j+1} E_j^T w - \mathcal{Q}_j w\lambda\right|\right| \\ &= \left|\left|Q_{j+1} B_{j+1} E_j^T w\right|\right| = \left|\left|B_{j+1} E_j^T w\right|\right| \\ &\equiv \left|\left|B_{j+1} w^{(p)}\right|\right|, \end{aligned} \tag{5.3}$$

where $w^{(p)}$ denotes the last p components of w. Note that the spectral norm is unitarily invariant, which is why the the block Q_{j+1} was dropped in the last equality. For the vector Lanczos, Eq. (5.3) reduces to

$$||Av - \lambda v|| = \left|\beta_{j+1} w^{(1)}\right|, \tag{5.4}$$

where $w^{(1)}$ is the last component of w. Eq. (5.3) provides a cheap measure since B_{j+1} is $p \times p$ and the number of arithmetic operations involved in an eigenvalue decomposition of T_j is independent of the dimension n of the large matrix A. In practice, it turns out that the last components of w are typically small, thus lending credence to the fact that μ and $\mathcal{Q}_j w$ provide a good approximation to an eigenpair of A.

The MATLAB implementations used for conducting the numerical tests throughout this chapter all employ the stopping criterion in Eq. (5.1) where the

residual norm is computed using Eq. (5.3). The algorithms were halted at regular intervals where Ritz pairs were computed from the current block tridiagonal matrix T and the criterion in Eq. (5.1) was subsequently checked for all the eigenvalues of interest.

Note that Eq. (5.1) includes both the eigenvector and the corresponding eigenvalue, but for some applications only the spectrum of A is required and it is then natural to ask if a particular value for tol can be translated into the accuracy of the computed eigenvalue. That is, if the true eigenvalue is denoted λ and the approximation computed by a Lanczos method is $\tilde{\lambda}$, the goal is to relate the absolute error $\epsilon_{\text{abs}} = |\lambda - \tilde{\lambda}|$ to tol in Eq. (5.1). Indeed, it is not difficult to show [10] that Eq. (5.3) implies that

$$|\lambda - \tilde{\lambda}| \leq \left|\left| A\tilde{v} - \tilde{\lambda}\tilde{v} \right|\right| = \left|\left| B_{j+1} w^{(p)} \right|\right|. \tag{5.5}$$

Thus, if it were possible to determine the right hand side of Eq. (5.5) to working-precision it would immediately follow from Eq. (5.5) that the computed eigenvalue was accurate to working-precision as well. Previous results [14] show however, that

$$\left|\left| A\tilde{v} - \tilde{\lambda}\tilde{v} \right|\right| \leq \epsilon_{\text{mach}} \, ||A|| \tag{5.6}$$

is the best that can be done. To illustrate this fact the Lanczos algorithm was used to find an approximation of the least eigenvalue $\tilde{\lambda}_1$ of a 5147×5147 matrix A with different values of the tolerance in Eq. (5.1). Fig. 5.2 relates these to the corresponding absolute error $|\lambda_1 - \tilde{\lambda}_1|$, shown on a logarithmic scale. The true eigenvalue λ_1 was computed to working precision using a QR-based eigensolver. From Fig. 5.2 it is evident that values of tol $\leq 10^{-4}$ yields an approximate eigenvalue which is correct up to the tenth or eleventh decimal place but no more. This is just a reflection of Eq. (5.6) which becomes apparent when evaluating the right hand side where $||A|| \, \epsilon_{\text{mach}} \approx 10^{-11}$.

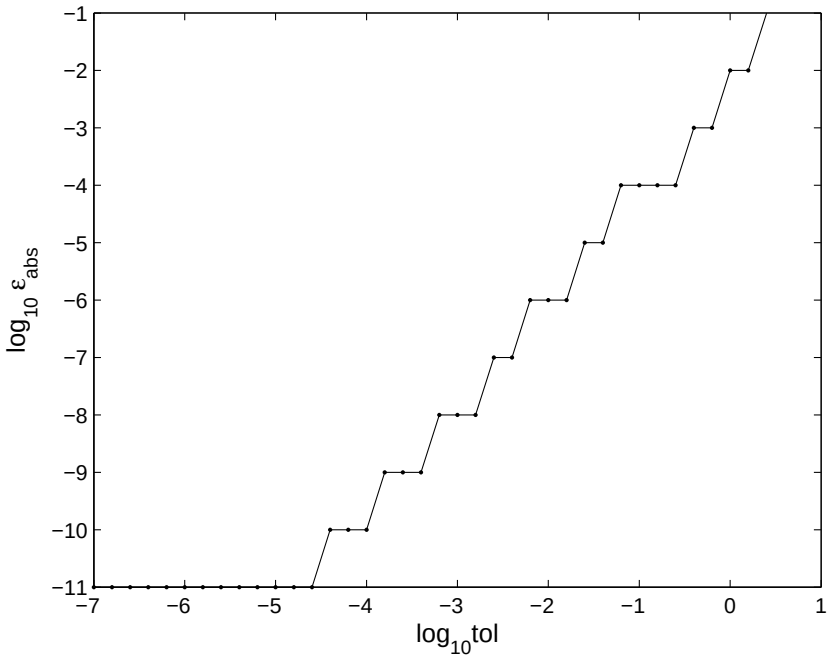


Figure 5.2. *The absolute error* $\epsilon_{\text{abs}} = |\lambda - \tilde{\lambda}|$ *of the computed eigenvalue* $\tilde{\lambda}$ *as a function of the tolerance used in Eq.* (5.1)*, shown with logarithmic scales.*

Following Eq. (5.6), a more appropriate stopping criterion would be

$$\left|\left|A\tilde{v} - \tilde{\lambda}\tilde{v}\right|\right| \leq \text{tol}\,||A||\,. \tag{5.7}$$

This definition of the user specified tolerance more accurately reflects the resulting number of correct digits in any computed eigenvalue, evidence for which can be supplied by re-examining Fig. 5.2. The tolerance as defined in Eq. (5.7) is now used instead of that in Eq. (5.1) yielding a more intuitive relationship between tolerance and accuracy shown in Fig. 5.3. As indicated by the figure, a particular order of magnitude of the tolerance now corresponds to roughly the same amount of accurate digits in the computed eigenvalue for approximately $10^{-10} < \text{tol} < 10^{-5}$. Above this interval, the condition in Eq. (5.7) is too lax and for values of $\text{tol} < 10^{-10}$ Eq. (5.6) limits the accuracy that can be achieved.

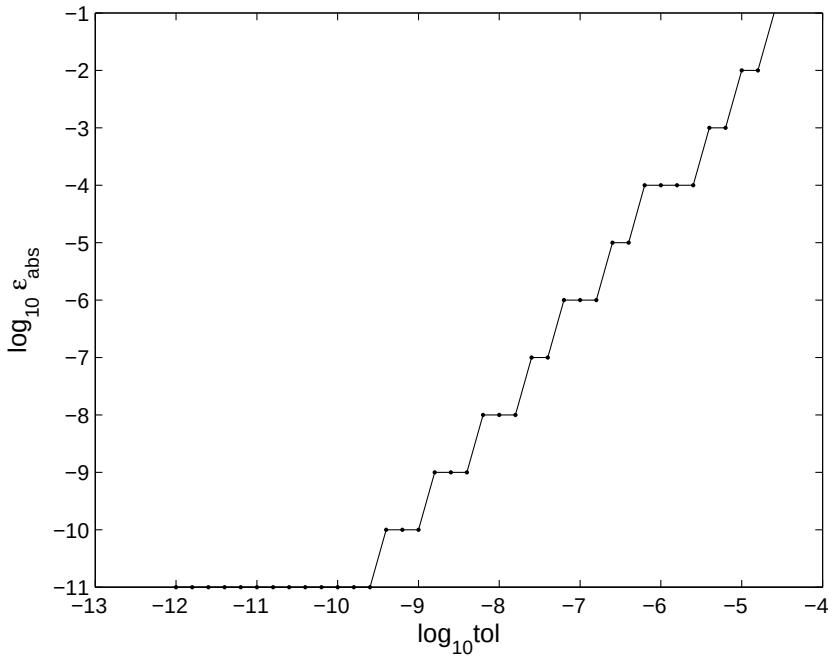


Figure 5.3. *The accuracy of the computed eigenvalue* $\tilde{\lambda}$ *measured by the absolute* $\epsilon_{\text{abs}} = |\lambda - \tilde{\lambda}|$ *as a function of the redefined tolerance used in Eq.* (5.7)*, shown with logarithmic scales. From this figure it is clear that the order of magnitude of the specified tolerance corresponds to roughly the amount accurate digits in the computed eigenvalue for values of* tol *with* $10^{-10} < \text{tol} < 10^{-5}$.

Indeed, this is the form of the stopping criterion usually encountered in Lanczos-based eigensolvers, see for example [13]. For the investigations in this chapter however, the original form of the criterion given in Eq. (5.1) will be used unless stated otherwise.

Furthermore, it has also been noted that $\tilde{\lambda} = \tilde{v}^T A \tilde{v}$ can be viewed as a Rayleigh quotient which in many cases obey the sharper estimate [10]

$$|\lambda - \tilde{\lambda}| \leq \frac{\left|\left| A\tilde{v} - \tilde{\lambda}\tilde{v} \right|\right|^2}{\gamma_i}, \tag{5.8}$$

where δ_i denotes the minimum gap between $\tilde{\lambda_i}$ and the rest of the spectrum of A. Assuming that the eigenvalues are reasonably well separated, the absolute error in Eq. (5.8) should thus fall off as roughly the square of the residual norm. This claim can be checked by taking the square of the tolerance in Fig. 5.2 yielding Fig. 5.4.

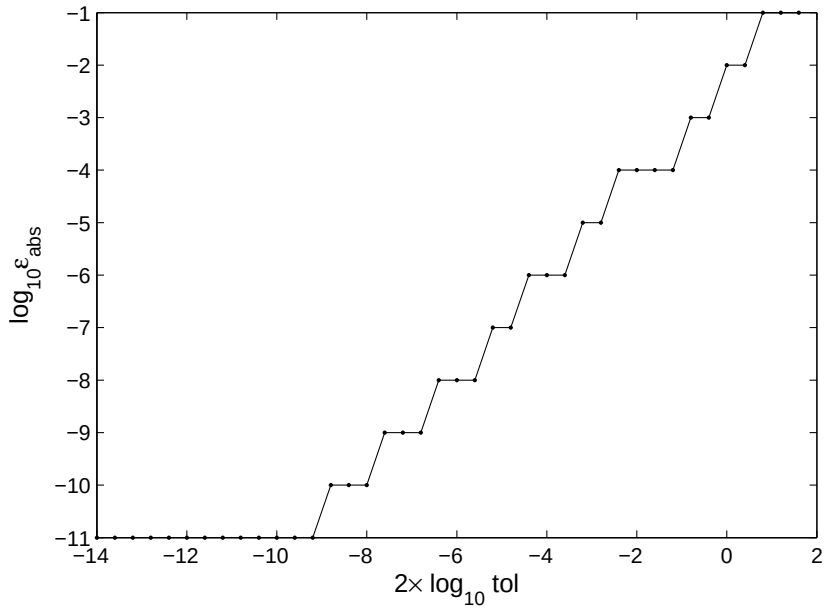


Figure 5.4. *The absolute error* $\epsilon_{\text{abs}} = |\lambda - \tilde{\lambda}|$ *of the computed eigenvalue* $\tilde{\lambda}$ *as a function of the square tolerance used in Eq.* (5.1)*, shown with logarithmic scales.*

Comparing Fig. 5.4 to Fig. 5.2 it is obvious that Eq. (5.8) gives a better estimate on the error, but in this particular case it is still an overestimation when comparing to Fig. 5.3.

As a concluding remark, note that in this section no mention of the speed of the convergence or the impact of the block size has been made. Readers are instead referred to [19] for rigorous theoretical bounds for the vector as well as the Block Lanczos method.

5.2 Comparison of the Lanczos and the Block Lanczos method

This section examines how block size affects the computational cost of obtaining eigenpairs with a given tolerance, where an eigenpair is defined as an eigenvalue with its associated eigenvector. The study is performed on a number of matrices depicting a system of 6 bosons, differing from each other in that they have different values of the cut-off energy $N_{\max}$. All of these matrices are generally denoted by A, but sometimes the matrix dimension n is also specifically defined.

As the dimension of the $n \times n$ matrix A grows large, the cost of each matrix multiplication increases as well. It is therefore necessary to find out if the use of blocks instead of vectors, which will allow each multiplication to produce more information, will increases the convergence enough to outweigh the increased cost of using several vectors instead of one.

Fig. 5.5 shows the required multiplications by A for a certain precision, as a function of the block size[2]. In this case randomly generated starting blocks were employed.

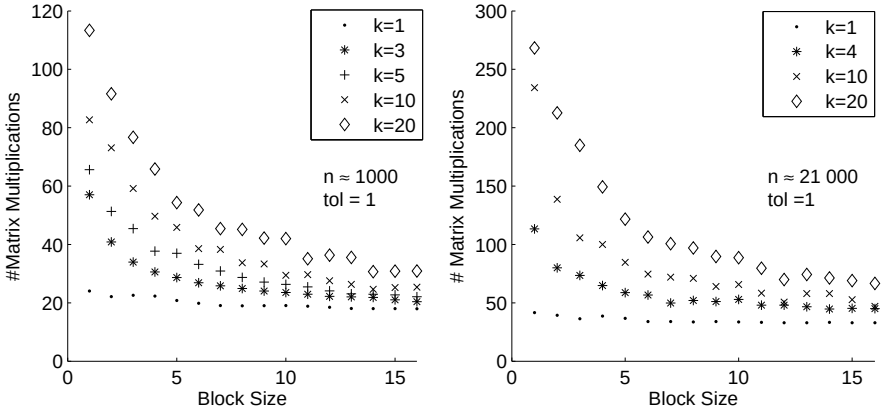


Figure 5.5. *The number of required multiplications by the matrix A to obtain a precision $\epsilon = 1$ from a set of random startvectors, versus the block size p. Each value is the mean value of 5 takes. k is the number of computed eigenpairs. Left: $k = 1,3,5,10,20$, $n \approx 1000$. Right: $k = 1,4,10,20$, $n \approx 21\,000$.*

From Fig. 5.5 it can be seen that larger blocks require less multiplications, and this holds for all values of the desired eigenpairs k in the figure. However, it comes at a price, since even though the number of multiplications

[2]Note that the regular Lanczos Method is equivalent to the Block Lanczos Method with a block size $p = 1$.

by A decreases, the storage cost of the Krylov space created does not. After s multiplications by the $n \times n$ matrix A and with a block size of p, the dimension of the created $\mathcal{Q}$ matrix is going to be $n \times sp$, and the matrix T will be of size $sp \times sp$. Especially $\mathcal{Q}$ will soon require a lot of storage space as p increases, as $n \gg sp$, in general. Another effect of the larger blocks is that the cost of reorthogonalization grows as the blocks of $\mathcal{Q}$ holds more vectors that needs to be held orthonormal to each other.

However, the great advantage of dealing with blocks stems from the fact that when a computer needs to access an element in a vector, it needs to collect the whole cache-line, in which that element lies, to its cache-memory.[3] By using blocks instead of vectors, elements of the same row in the block can be stored in the same cache-line. This makes it possible to access the whole block row more or less with the same effort as for one element, as long the block size is smaller than eight. Since it is the transfer from the memory to the cache-memory that is limiting, the extra multiplications are much cheaper than they would be if the vectors of the blocks were collected one by one.

Not all block sizes are practical for computations however. For instance, a block size of $p = 9$ is terrible in terms of cost, as it means that two cache-lines need to be collected for each row of the block and seven positions of the second line are unused. If a block size of $p = 16$ instead is used the cache will still collect two cache-lines but in this case it will use all the data. A block size of $p = 1$,2 or 4 is advantageous as it allows additional block rows to be stored in the same cache-line, without wasting any space.

To decide the optimal block size, the average time for a matrix-block multiplication was measured and normalized according to the time it takes to compute a regular matrix-vector multiplication. The measured data can be seen in Table 5.1, [20]. It can be seen that for instance with a block size $p = 2$, the matrix multiplication takes 1.1 times the time of a regular multiplication, with $p = 1$.

[3] In current hardware, these cache-lines usually have a capacity of 64 bytes, i.e. eight elements of size *double* (8 bytes).

Table 5.1. *The relative computational time for matrix-vector multiplication with different block sizes.*

Blockwidth	Computational time
1	1
2	1.1
3	1.13
4	1.25
8	1.86
16	3.61
32	8.65

By weighting the factors from Table 5.1 with the required number of multiplications with A for a certain block size, as seen in Fig. 5.5, this yields a figure of how fast the eigenpairs can be computed as a function of block size p. This can be seen in Fig. 5.6. Note that one unit of time is the time it takes to multiply a regular vector with A once. Values for the computational time for block sizes $p = 5-7$ and $p = 9-15$ were approximated to the same values as for $p = 8$ and 16 respectively, which are found in Table 5.1. This since the entire cache-line must be collected from memory in both of those cases.

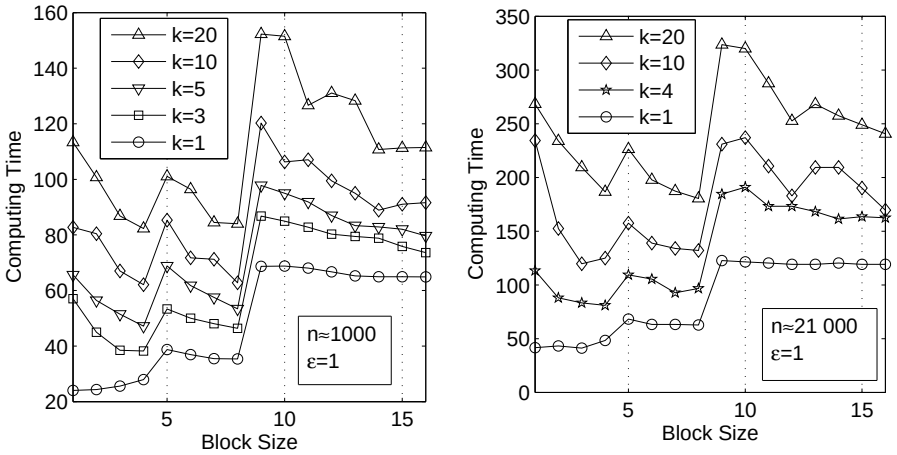


Figure 5.6. *The time taken to compute eigenpairs of the matrix A with a precision of* tol $= 1$ *and a set of random startvectors, versus the block size p. The computing time is in unit of the time it takes to multiply a regular vector with A once. Each value is the mean value of* 5 *runs. k is the number of computed eigenpairs. Left: $k =$* 1,3,5,10,20*, $n \approx 1000$. Right: $k =$* 1,4,10,20*, $n \approx 21\,000$.*

From Fig. 5.6 it is clear that if more than one eigenpair is desired, $p = 4$ or 8 is the ideal block size using a random starting vector. If the desired number of eigenpairs is $k < 10$, a block size of $p = 4$ is the natural choice, and with

values of $k > 10$ it is more or less a dead race. If the computing time of a block size $p = 4$ and $p = 8$ is equivalent at this point it could be better to choose the former, as this would keep the Krylov space dimension down, making the reorthogonalizations less expensive. These advices so far only apply to the case of a random starting block. The effects of other starting blocks are studied in Section 5.10.

5.3 Strategies for restoring orthogonality

Section 4.4 showed that without modification, the mutual orthogonality between Lanczos blocks (or vectors) is gradually lost as the dimension of the Krylov space increases. This eventually leads to the appearance of spurious or duplicate eigenvalues which are hard to separate from the true eigenvalues. In order to avoid this scenario several schemes for restoring orthogonality among the Lanczos blocks were proposed. It was also noted that any form of reorthogonalization invariably comes at a price. In particular, previously generated Lanczos blocks are often required to be kept in main memory so as to be readily available for reorthogonalization. Furthermore, since these blocks are composed of n-vectors, the reorthogonalizations themselves introduce a significant number of arithmetic operations that would not otherwise be present. Any practical Lanczos method should therefore strive to keep such reorthogonalizations at a minimum, both in terms of the frequency with which they are performed and the number of blocks which are being operated on.

The purpose of this section is a numerical investigation of what is perhaps the most straightforward way of reducing the number of reorthogonalizations needed, namely Partial Reorthogonalization (PRO) which was discussed at length in Section 4.4. As was briefly indicated in Section 5.2, the Block Lanczos algorithm seems to be the preferable choice when dealing with large-scale eigenproblems. Therefore, the following investigations are based on a basic implementation of the Block Lanczos algorihtm as described in Section 4.2.1, Algorithm 2.

To begin with, the basic PRO-scheme outlined in Section 4.4.3, Algorithms 6 and 7, will be compared to an otherwise identical implementation using a full reorthogonalization (FRO) (Section 4.4.2, Algorithm 5). Next, the scheme for modified partial reorthogonalization (MPRO), Algorithm 8, will be compared to the basic PRO-scheme to see whether the increased complexity of the MPRO-scheme will yield fewer orthogonalizations.

5.3.1 Properties of partial reorthogonalization

Consider the Block Lanczos algorithm with partial reorthogonalization; it is impossible to tell from Algorithms 6 and 7 the amount by which the number of orthogonalizations can be reduced when comparing to a Block Lanczos solver with full reorthogonalization. A related quantity of interest is the number of reorthogonalization steps, or recalls. This is simply the number of times a reorthogonalization happens and is equal to the number of times that the reorthgonalization criterion

$$\max_x \omega_{j+1k} \geq \sqrt{\epsilon} \tag{5.9}$$

is violated, see Section 4.4.3 for a reminder.

To get an idea of the numbers involved two instances of the Block Lanczos algorithm, one using PRO and the other FRO, was run 10 times each on a matrix A of order 4966×4966. Both instances used the same randomly generated starting block for each run and were asked to compute a total number of six eigenvalues with a block size $p = 6$ and the tolerance set to 10^{-3}. The results are summarized in Table 5.2, where the second column is the average number of steps required for convergence and the third column is the average number of orthogonalizations performed in a blockwise sense, i.e. the orthogonalization of a block Q_2 against Q_1 counts as a single orthogonalization. The last column is the average number of recalls.

Table 5.2. *A comparison of partial reorthogonalization and full reorthogonalization.*

Scheme	Steps	Orthogonalizations	Recalls
PRO	111	3342	28
FRO	111	6229	111

Table 5.2 indicates that the Block Lanczos with PRO used on average 53% as many orthogonalizations as the Block Lanczos using FRO and only 25% as many recalls, which is a considerable improvement. Furthermore, the second column in Table 5.2 indicates that the convergence was equally fast in both cases. Note that the dimension of the corresponding Krylov space does not equal the number of iterations but rather p times this number.

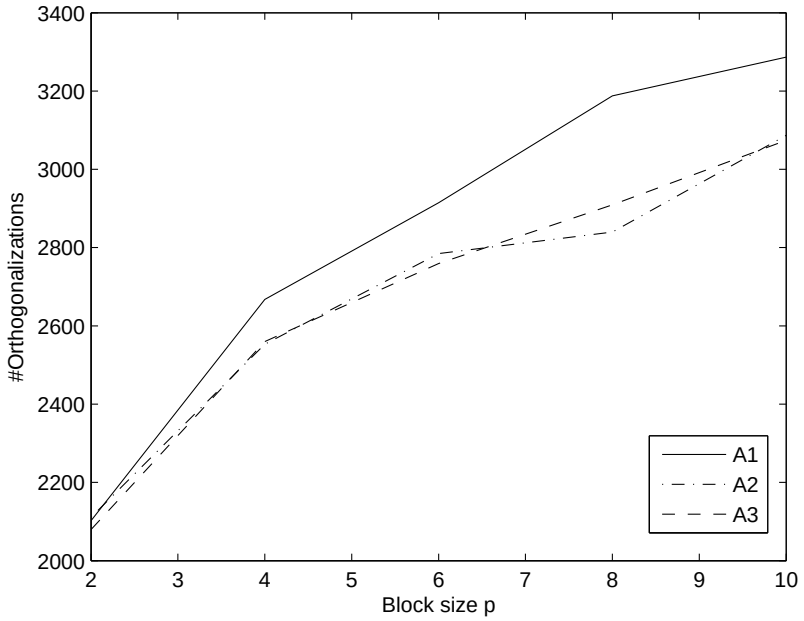
In practical applications, the dimension of the matrix is usually much larger and the block size used may range somewhere between $p = 2 - 10$ (see Section 5.2 for an analysis of the impact of the block size on convergence and more). Since orthogonalizations can quickly become a bottleneck for large-scale computations it is important to know how the numbers from Table 5.2

scale with the block size, dimension of the Krylov space and order of the matrix.

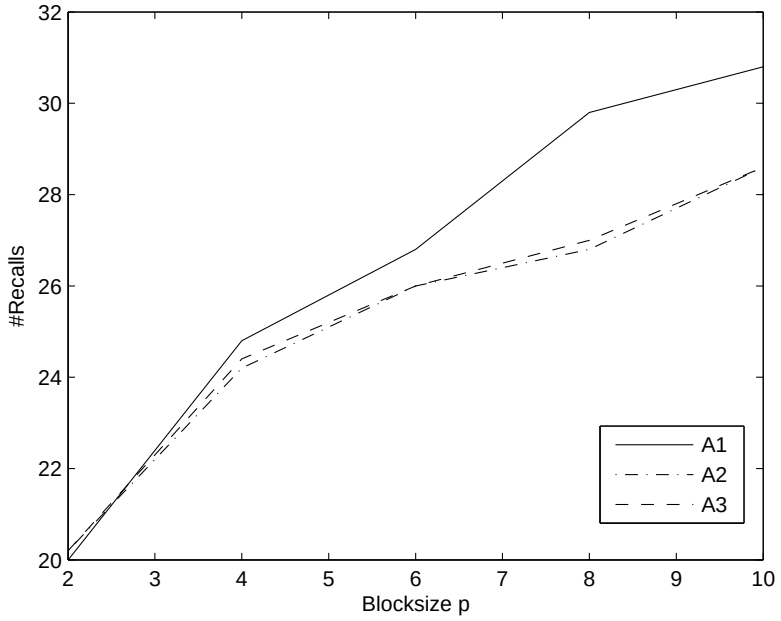
For this purpose, the Block Lanczos with PRO was run for a total of 100 steps on three different test matrices A_1, A_2 and A_3 with sizes 5147×5147, 25474×25474 and 119240×119240. The range of the block size was $p = 2, 4, 6, 8, 10$. The results are displayed in Fig. 5.7, where the number of orthogonalizations and recalls are both plotted as functions of the block size p. The counts were averaged over a total of five runs for each value of p. New starting blocks were randomly generated for each value of p and for each matrix.

From Fig. 5.7 it is evident that both the frequency of the reorthogonalizations, i.e. the number of recalls divided by the total number of steps, and the amount of actual orthogonalizations is insensitive to the order n of the matrix for values of n up to at least 10^5. Furthermore, the frequency and number of orthogonalizations both increase with the block size. In fact, the results could both have been predicted from the simulation of the loss of orthogonality in Algorithm 6. Here, the only quantity that depends on n is ϵ_s, which only plays a minor role in the propagation of the losses, whereas the block size p determines the size of all the small matrices and hence affects the corresponding spectral norms.

Note that in Fig. 5.7 the growth of the number of orthogonalizations falls of somewhat for the higher values of p. This behavior can be examined more closely by focusing on one of the matrices above. For A_1, two expanded plots of number of orthogonalizations and recalls as functions of block size is shown in Fig. 5.8. Here, number of orthogonalizations/recalls is displayed as a fraction of the corresponding number made by the Block Lanczos with FRO. These figures thus give an estimate of how the benefits of PRO diminishes as the block size increases. The result is actually more general than the experiment suggests since no real correlation between the number of orthogonalizations and the order of the matrix was found for $n \sim 10^5$.

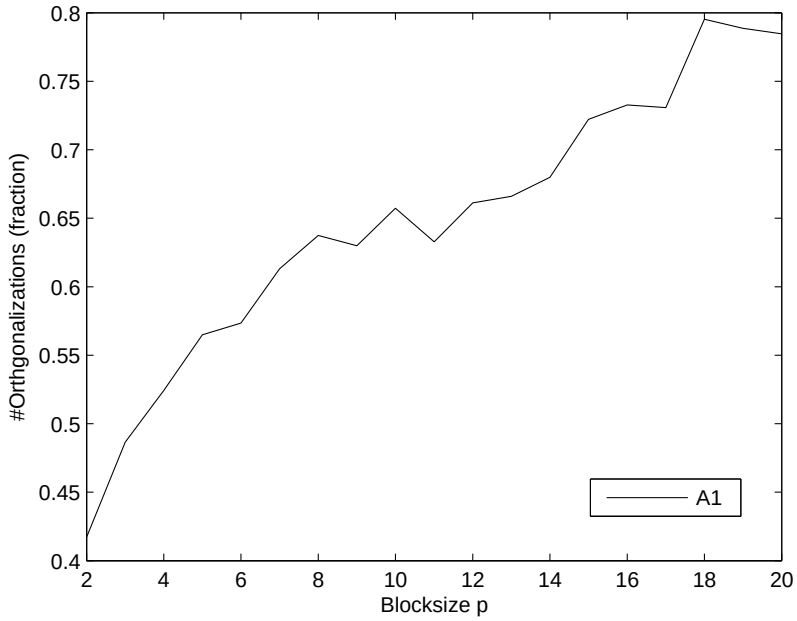


(a) The average number of orthogonalizations versus the block size p.

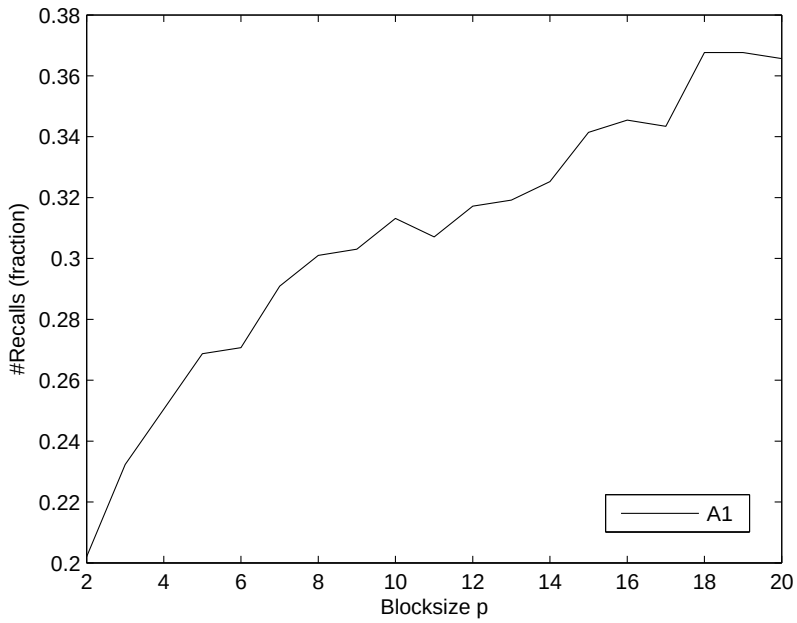


(b) The average number of recalls versus the block size p.

Figure 5.7. *The average number of (a) orthogonalizations and (b) recalls computed by the Block Lanczos with PRO shown against block size p, for the three matrices A_1, A_2 and A_3 of different order.*



(a) The average number of orthogonalizations, as a fraction, versus the block size p.

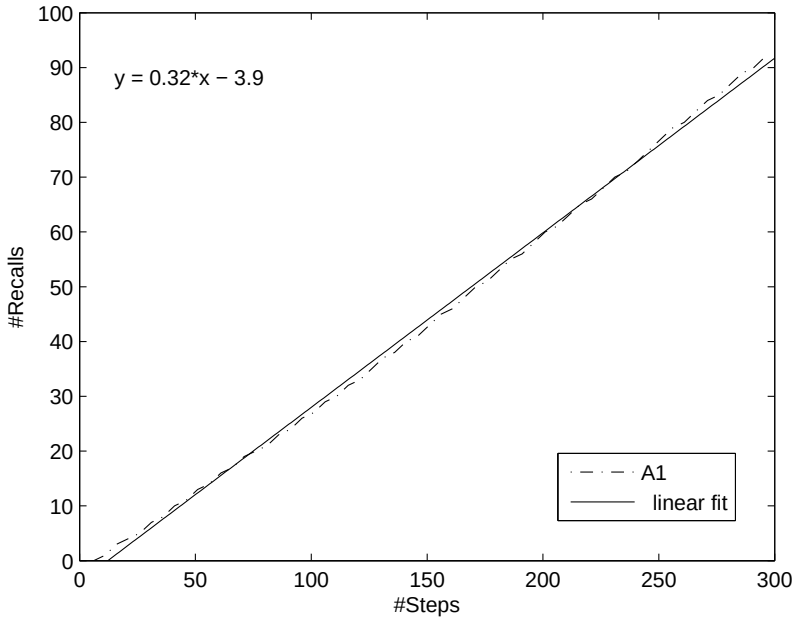


(b) The average number of recalls, as a fraction, versus the block size p.

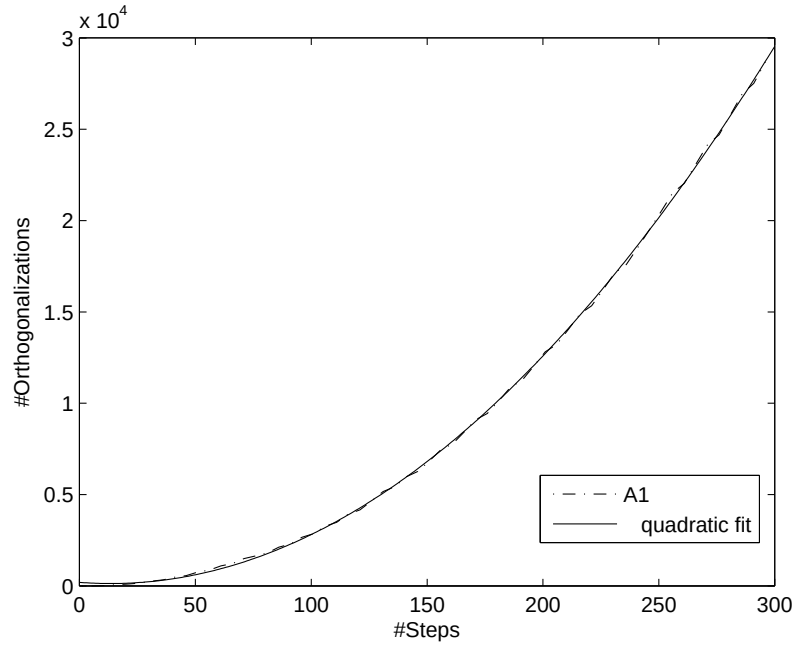
Figure 5.8. *The average number of (a) orthogonalizations and (b) recalls computed using the PRO-scheme, displayed as fractions of the corresponding numbers for FRO, versus block size p for the matrix* A_1.

The fluctuations in Fig. 5.8 are mainly due to the random selection of new starting blocks. From Fig. 5.8 (a) it is possible to conclude that in terms of the pure number of orthogonalizations, PRO is nearly as bad as FRO for block sizes around $p = 20$ where the fraction is about 80%. However, in terms of the number of recalls, there is still a significant improvement since only about 36% as many recalls is done using PRO. The reason for the discrepancy between these two quantities is that during a step of reorthogonalization in PRO, the two most recent blocks are orthogonalized against all their predecessors while in FRO only the most recent block is used.

Lastly, consider the number of orthogonalizations as a function of the number of steps or equivalently the dimension of the Krylov space since the two are proportional. Since no s-dependent quantity is involved in the updating, Eq. (4.50), of the simulation in Algorithm 6 and the PRO-scheme is designed to maintain semi-orthogonality among the Lanczos blocks, the frequency of the recalls should remain constant as the number of steps increases. Numerical results show to this is indeed the case, see Fig. 5.9 (a). Here the number of recalls made by the Block Lanczos with PRO was plotted for the matrix A_1 as function of step length s, displayed as a dashed line. The full line is a linear fit to this plot with an approximate gradient $k = 0.32$. Thus the number of recalls is expected to increase by a factor 0.32 for each increment of the step length or equivalently, a step of PRO is done approximately every third step. A plot of the number of orthogonalizations as a function of number of s is shown in Fig. 5.9 (b). The expected behaviour here is a quadratic increase since the amount of orthogonalizations is proportional to both the number of available blocks, which equals the step s, and the number of recalls which was also shown to be proportional to s above. Hence, the number of orthogonalizations should be $\sim s^2$ which is confirmed by the quadratic fit in Fig. 5.9 (b).



(a) The average number of recalls (dashed line) versus the step length s.



(b) The average number of orthogonalizations (dashed line) versus the block size p.

Figure 5.9. *The average number of (a) recalls and (b) orthogonalizations computed using the PRO-scheme, as a function of the number of steps s, for the matrix A_1. The block size was set to $p = 6$ The full line represents (a) a linear fit and (b) a quadratic fit to the curve.*

5.3.2 Comparison to modified partial reorthogonalization

In an attempt to further minimize the number of orthogonalizations/recalls the MPRO-scheme of Section 4.4.4 will be compared to the PRO-scheme which was thoroughly investigated in the last section. The crucial difference between MPRO and simple PRO is the number of Lanczos blocks which are selected for reorthogonalization. In the MPRO-scheme these are grouped into batches, where the size of the batch is determined by the free parameter η, see Algorithm 8. Since MPRO also uses the ω-recurrence (Algorithm 6) to simulate the loss of orthogonality its dependency on the block size, step length and matrix order is essentially the same but the actual number of orthogonalizations/recalls will depend on η.

In practice, it is impossible to make theoretical predictions concerning the optimal value of η. Instead, an experimental approach is adopted where the values of η will range over an appropriately chosen interval for a few selected test matrices. The resulting data will then be used to determine a minimum. By Algorithms 6 and 7, the level of orthogonality between blocks is bounded by ϵ_s and $\sqrt{\epsilon}$, hence values of η should lie in the interval $[\epsilon_s, \sqrt{\epsilon}]$.

The Block Lanczos with MPRO was run for 60 steps with the block size $p = 4$ on the 5147×5147 matrix A_1 from the previous Section 5.3.1. The bounds of the level of orthogonality were given by $\epsilon_s = 6.37\times10^{-14}$ and $\sqrt{\epsilon} = 1.49\times10^{-8}$. The resulting number of orthogonalizations and recalls are summarized in Table 5.3 for their corresponding values of η. The last row indicates the corresponding values for PRO.

Table 5.3. *A comparison of MPRO and PRO for different values of the batch size bound η. The first six rows are the results for the MPRO-scheme while the last row gives the corresponding values for PRO.*

η	Recalls	Orthogonalizations
$10^{-1}\sqrt{\epsilon}$	47	734
$10^{-2}\sqrt{\epsilon}$	29	728
$10^{-3}\sqrt{\epsilon}$	23	705
$10^{-4}\sqrt{\epsilon}$	22	705
$10^{-5}\sqrt{\epsilon}$	22	705
$10^{-6}\sqrt{\epsilon}$	22	705
PRO	14	1001

From Table 5.3 it is clear that values of η in the range $10^{-6}\sqrt{\epsilon}$ to $10^{-4}\sqrt{\epsilon}$ yields the fewest number of recalls as well as orthogonalization. The fact that they remain constant for $\eta < 10^{-4}\sqrt{\epsilon}$ is due to the neighborhood of the first offending k detected by Algorithm 8 having grown large enough to include all the previously generated Lanczos blocks and hence no further values of

k will be found. Comparing these results to the last row containing the corresponding numbers for PRO, it is evident that MRPO reduces the amount of orthogonalizations while the number of recalls increases. This last fact is due to the difference in implementations of MPRO and PRO, as the particular PRO-scheme described in Section 4.4.3 included local orthogonalization as well as a reorthogonalization of the two most recently computed blocks rather than only the most recent block as in MPRO. Hence, at least twice as many blocks are chosen for each recall operation in PRO and because of this the recalls occur less frequently.

To see if these numbers scale, a similar test was done for the 25474×25474 matrix A_2 from Section 5.3.1. The step length was set to $s = 100$ and a block size of $p = 8$ was used. The results are shown in Table 5.4.

Table 5.4. *A comparison of MPRO and PRO for different values of the batch size bound η. The first six rows are the results for the MPRO-scheme while the last row gives the corresponding values for PRO.*

η	Recalls	Orthogonalizations
$10^{-1}\sqrt{\epsilon}$	76	1918
$10^{-2}\sqrt{\epsilon}$	63	2085
$10^{-3}\sqrt{\epsilon}$	40	2114
$10^{-4}\sqrt{\epsilon}$	40	2122
$10^{-5}\sqrt{\epsilon}$	40	2122
$10^{-6}\sqrt{\epsilon}$	40	2122
PRO	24	2677

In this case, a larger bound $\eta = 10^{-1}\sqrt{\epsilon}$ on the size of the batches seems to yield fewer orthogonalizations while keeping the number of recalls at a maximum. Yet the difference in the number of orthogonalizations is comparatively small between different values of η and while this is clearly an improvement on the 2677 orthogonalizations computed by the original PRO, as indicated in the last row, it is unclear whether MPRO is preferable since the number of recalls seems to increase as well. There is one caveat when comparing the number of recalls for different values of η: The amount of work is proportional to the number of recalls times the batch size, but the latter is determined by η. Hence, the amount of work required for scanning and retrieving old Lanczos blocks cannot be directly inferred from the number of recalls alone.

5.4 Starting vectors

In this section a study follows of the behaviour of the Lanczos Algorithm when the starting blocks are created by approximations of the desired eigenvectors. It is essentially divided into two separate parts. One that will deal with how to acquire the approximations, and the other with how to best apply them. Should for example each approximation become one vector in the starting block, or should each vector in the starting block be a combination of many approximations? Before such a discussion may begin, some tools are needed in order to measure the quality of an approximation. Since the essence of this really is how well a vector fulfills its role as an eigenvector, an easy way to evaluate that would be to use the same tolerance measure that was earlier used as a stop criteria for the Lanczos Algorithm:

$$\epsilon = ||A\tilde{v} - \tilde{\lambda}\tilde{v}||_2 \tag{5.10}$$

where $\tilde{v}$ is the eigenvector approximation, and $\tilde{\lambda}$ is an eigenvalue approximation.

5.4.1 Effects of well chosen starting vectors

There are some different strategies for building start blocks from a given set of eigenvalue approximations. Suppose that eight eigenpairs are wanted. As shown in Section 5.2 the optimal block size p, at least with a block of random starting vectors, turned out to be about $p = 4$ if eight eigenpairs were desired. Therefore it is interesting to examine if it still is preferable to use a block size $p = 4$ – with a combination of two of the approximations in each vector – or if it is better to use block size $p = 8$ even though every matrix multiplication would be more time consuming.

Let us begin by looking at two approximate eigenvectors, $\tilde{v_1}$ and $\tilde{v_2}$, and two desired eigenpairs. From Section 5.2 it seems as a block size of $p = 2$ would be optimal for this number of eigenpairs. To investigate this, the number of matrix multiplications needed to gain the eigenpairs, with a tolerance of 10^{-10} was computed, with different quality of the approximations in the sense of Eq. (5.10). The different strategies to utilize the approximations were:

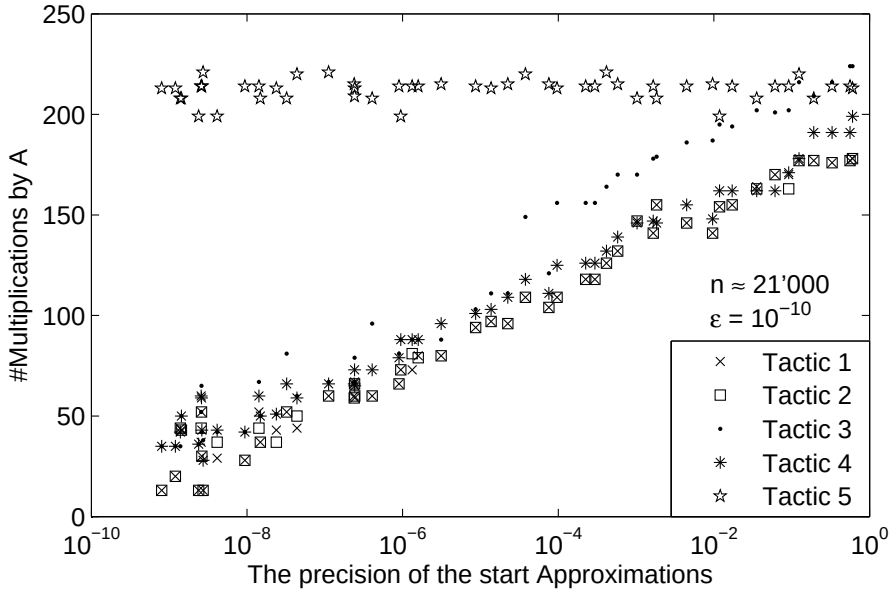


Figure 5.10. *The number of block-matrix multiplications needed to compute the two first eigenpairs with a tolerance of $\epsilon = 10^{-10}$, as a function of how well chosen the starting vectors were, and for a matrix of dimension n. The different curves corresponds to different tactics of using these approximations which can be seen in Table 5.5.*

Table 5.5. *Generated starting blocks.*

Tactic	**Vector 1**	**Vector 2**
1	$\tilde{v_1}$	$\tilde{v_2}$
2	$\frac{\tilde{v_1}+\tilde{v_2}}{\sqrt{2}}$	$\frac{\tilde{v_1}-\tilde{v_2}}{\sqrt{2}}$
3	$\frac{\tilde{v_1}+\tilde{v_2}}{\sqrt{2}}$	*random*
4	$\frac{\tilde{v_1}+\tilde{v_2}}{\sqrt{2}}$	–
5	*random*	*random*

The result of the investigation may be seen in Fig. 5.10. It is interesting that of all the curves in Fig. 5.10 that employ the approximations, the curve of tactic 3 is the least successfully. How come that for instance tactic 4 is better? Both tactic 3 and 4 have their first starting vector in common, but as tactic 4 only uses a block size $p = 1$, tactic 3 has one additional vector that is randomly created. One explanation of the difference might be that as one random vector is introduced, components of undesired eigenvectors are introduced as well. This lowers the overall approximation precision, and thus gives the algorithm a less favourable starting point. Thus, it seems like

one vector, containing as few unwanted components as possible, is better than two vectors with the same data plus some unwanted data. What is also evident considering tactic 1 and 2 is that it does not matter if the start vectors are mixed or held apart in the start approximation, as long as they store the same amount of information. It also seems as if tactic 4 with only one block is almost as good as tactic 1 or 2. If the extra time taken for each matrix multiplication by a block of size $p = 2$ would have been taken into account there would be no difference at all. Even though this was the result of but one matrix, investigations done with other matrices gave similar results. Some general behaviour for tactic 1,2 and 4 seems to be that if the starting vectors are as good as the stop criteria – in this case 10^{-10} – almost no multiplications are needed, and that if the approximations have a tolerance of > 1, they do not give any advantage over a random block of size $p = 2$.

To continue, it was investigated how the results hold for four eigenvalues. The tactics that was employed were:

Table 5.6. *Generated starting blocks.*

Tactic	**Vector 1**	**Vector 2**	**Vector 3**	**Vector 4**
1	$\tilde{v_1}$	$\tilde{v_2}$	$\tilde{v_3}$	$\tilde{v_4}$
2	$\frac{\tilde{v_1}+\tilde{v_2}}{\sqrt{2}}$	$\frac{\tilde{v_3}+\tilde{v_4}}{\sqrt{2}}$	—	—
3	$\frac{\tilde{v_1}+\tilde{v_2}+\tilde{v_3}+\tilde{v_4}}{\sqrt{4}}$	—	—	—

The acquired results can be seen in Fig. 5.11 and Fig. 5.12. So far it seems as if a block size equal to the number of approximations is to prefer, but what happens if a greater number of eigenpairs are requested? For eight eigenpairs, Section 5.2 suggested that $p = 4$ or 8 in block size would do equally well, therefore we examined if this holds when using approximations as starting vectors as well. Tactic 1 this time is a block with $p = 8$ and one approximation in each vector, while Tactic 2 uses a block with $p = 4$ and two approximations in each. Fig. 5.13 shows the result. From these figures it can be concluded that the starting approximations must have a tolerance < 10 in order for them to be of any use at all. If only approximations are used it seems as if the recommendations of block sizes in Section 5.2 holds, with a possible exception of the case when the approximations are already quite close to the desired precision. It could then be better to use a block size equal to the number of approximations.

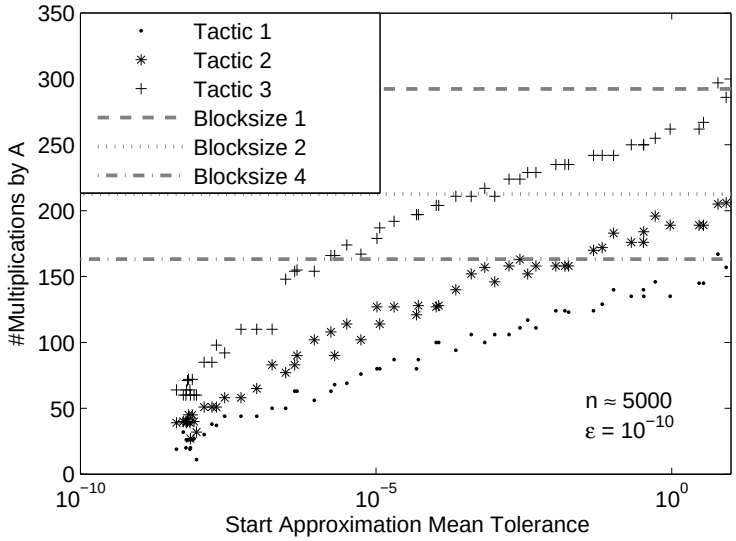


Figure 5.11. *The number of matrix multiplications needed to compute the four first eigenpairs with a tolerance of* $\epsilon = 10^{-10}$*, as a function of how well chosen the starting approximations were. The dashed lines marks the needed multiplications from a random start for block sizes* 1,2 *and* 4*. The matrix used was of dimension* 5000*.*

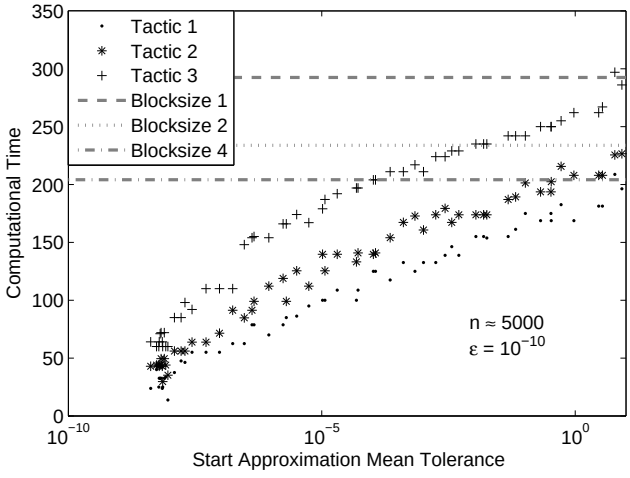


Figure 5.12. *The needed time to reach a tolerance of* 10^{-10}*, were one unit of time is the time it takes to multiply a regular vector with the matrix once, as a function of the tolerance of the input approximations.*

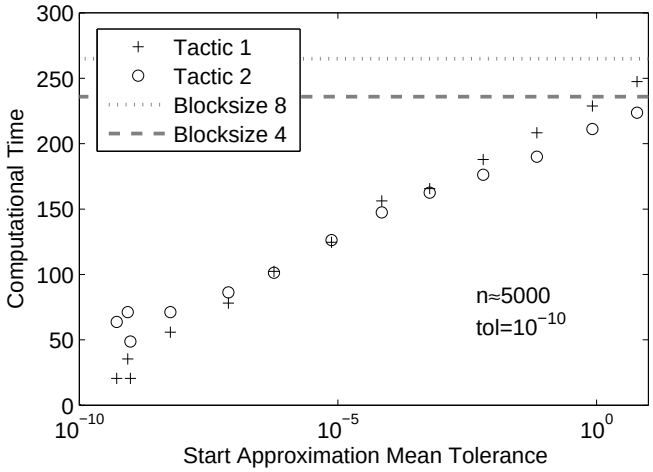


Figure 5.13. *The number of matrix multiplications needed to compute the eight first eigenpairs with a tolerance of* $\epsilon = 10^{-10}$*, as a function of how well chosen the starting approximations were in general. The dashed lines marks the needed multiplications from a random start for block sizes* 4 *and* 8*. The matrix used was of dimension* 5000*.*

5.4.2 Basic restarting schemes

Now when it is a bit clearer how to best put the approximations to use, some method is needed to compute them. The most straightforward way to construct accurate approximations to eigenpairs is to run the Lanczos algorithm with either a lower tolerance than the tolerance in the end requested, or with a Krylov space dimension already set in advance. The first method has the strength that the precision of the approximation is as good as desired, while the second option is advantageous in the sense that the needed data storage space is known in advance, but with the risk that the approximation is not good enough to give any advantages. However, both of these methods are of little use if the higher goal is to bring down the needed number of matrix multiplication. Fig. 5.14 shows the convergence of the fourth eigenpair of the same matrix as in Fig. 5.11.

The problem is that to acquire the desired precision of the previous figure it takes fewer iterations to simply continue in the first run, than it takes to create a start approximation and start over. The only time one thus would want to use this method to create start approximations is when the sizes of Q and T need to be small. The reorthogonalization costs of Q grows large as the dimension increases, and for smaller matrices this effect might as well be the dominating time consumer, which is why the restarted Lanczos is so widely used. There is however a special case when a restarted Lanczos could be attractive for computations of larger matrices. That is if the eigenpairs

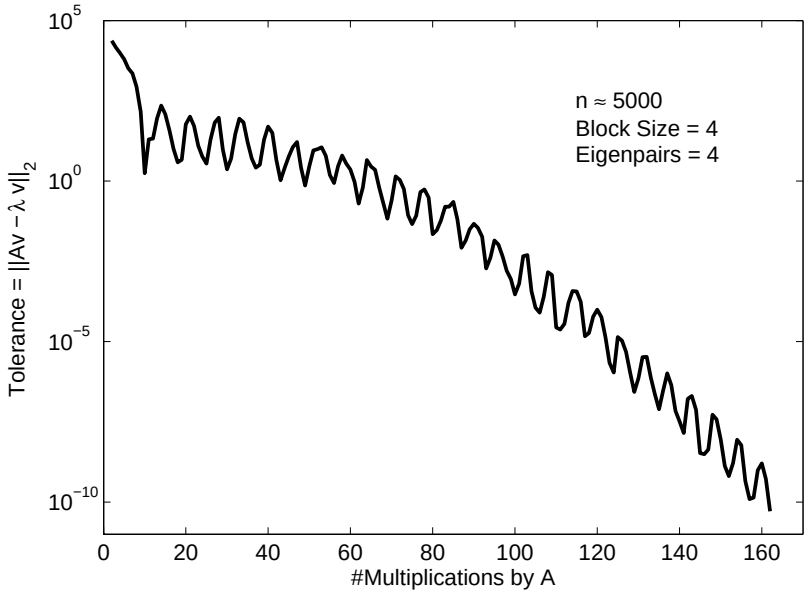


Figure 5.14. *This graph shows the convergence of the fourth eigenpair of the same matrix as in Fig. 5.11, from a random starting block of size* 4*.*

are collected in turns. If for example eight eigenvalues are needed, but only the first four pairs are approximated for some reason, then the Lanczos algorithm can find these first four pairs to the desired precision. As it does that, it will inevitable approximate the closest following eigenpairs as well, so when the first four eigenpairs has converged, the approximations of the following four could be used to form a start block of the next run[4].

5.4.3 Computations with reduced precision

Another general approach that is closely connected to the basic restart method is to decrease the floating point precision of the matrix for the first run when the approximations are created. A matrix stored in single (sp) instead of double precision (dp) only requires half the storage space. As the computational cost of the first run is less for sp matrices, it is not a problem if more matrix multiplications are needed in total, as long as the needed multiplications in the final run may be decreased.

Fig. 5.15 shows how good tolerance an eigenvector of a single precision matrix have compared to the double precision matrix, depending on how accurate it is computed for the sp matrix. It seems as if the approximations to the sp matrix at best have a precision of 10^{-4}, to the dp-matrix. A result

[4]This method would require that the earlier eigenvectors were forbidden.

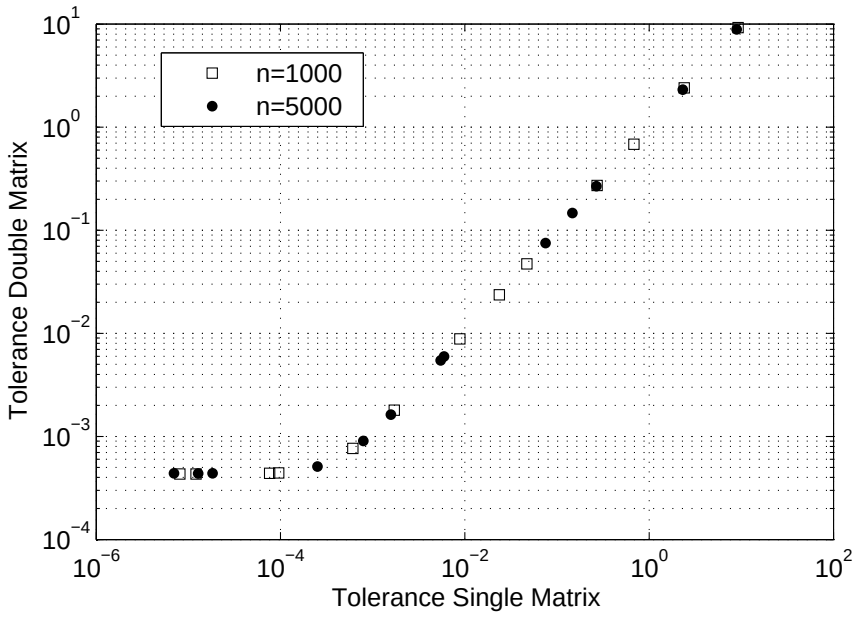


Figure 5.15. *This graph shows what tolerance an eigenvector to the single matrix has to the double matrix, compared to its precision to the single matrix. Two matrices are compared of dimension* 1000 *and* 5000*.*

that seems to be independent of the dimension of the matrix. The condition of the tolerance is reached when the eigenvector of the sp matrix has got a precision of 10^{-4}. To the matrix of dimension 5000 that was examined in Fig. 5.11 as well, this means that around 100 multiplications by the sp matrix gives four start approximations to the dp matrix of a tolerance 10^{-4}, which in turn, according to Fig. 5.11, means that just 100 matrix multiplications suffice to give an eigenpair accuracy of the four first eigenpairs of 10^{-10}. This requires in total 200 matrix multiplications to reach that tolerance level, in comparison to 160 that would be needed if they all were collected in one run of the dp matrix. However, the 100 multiplications of the single matrix is cheaper, and depending on how much cheaper, one strategy could be preferable over the other. What is clear however, is that if a tolerance $> 10^{-4}$ is acceptable, there is no need to use the dp matrix at all.

If one is only interested in the eigenvalues there is another trick. The dp and the sp matrices are slightly different and therefore their eigenvalues differ as well. In the example of the 5000×5000 matrix the difference between the eigenvalues of the sp and the dp matrices are in the scale of 10^{-5}. But a better eigenvalue can easily be approximated from the eigenvector approximation. Since this approximation is so close to the true eigenvector of the dp matrix, it will rescale approximately like the true eigenvector if multiplied

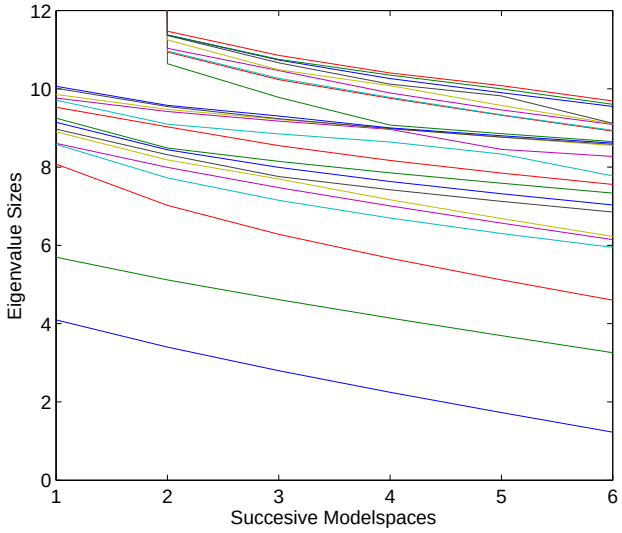


Figure 5.16. *This graph shows how the* 24 *first eigenpairs changes between the* 6 *first model spaces. The dimension of the successive matrices are:* $n_1 = 63$*,* $n_2 = 384$*,* $n_3 = 2065$*,* $n_4 = 9817$*,* $n_5 = 41\,787$ *and* $n_6 = 161\,032$*.*

by the dp matrix. By taking the norm of this multiplication an eigenvalue approximation is acquired that is only 10^{-8} away from the true eigenvalue. So by only using the dp matrix once it was possible to increase the accuracy of the eigenvalue approximation by a thousand times.

5.4.4 Approximations acquired from smaller model spaces

This last strategy is not general but highly dependent on how the matrices are formed. It arises from the fact that the Hamiltonian needs to be cut off at some energy level, and depending on where, different model spaces arise. The higher energy levels that are allowed, the better the approximation is of the true system. In this particular case when the bosons try to find the lowest possible energy configuration this means that the more states that are available, the more opportunities there are for new mixes of the Slater determinant building blocks, and this could ultimately lower the energy of the system. Therefore each eigenvalue found using a finite model space is slightly higher than the true value, even though the eigenvalues converge towards the true value as the cut-off level increases. Fig. 5.16 shows how the eigenvalues for successive model spaces changes. The fact that smaller model spaces approximate larger ones opens up for an opportunity to create start vectors in smaller spaces, and scale them for the larger ones, as all states of the smaller space also exist in the larger. The problem is however

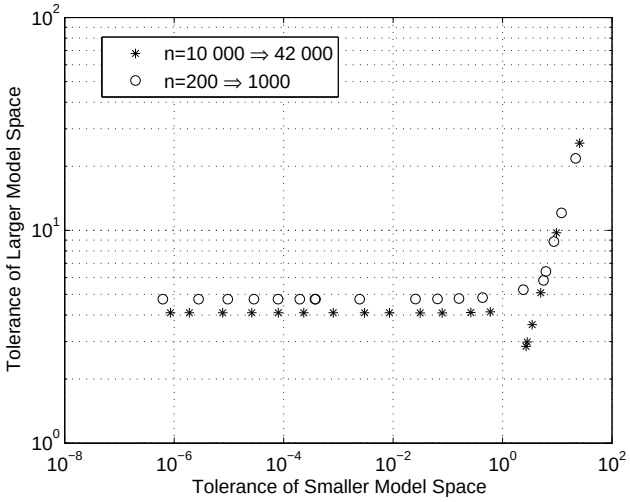


Figure 5.17. *This graph shows what tolerance an eigenvector to the smaller model space matrix have to the succeeding model space matrix, compared to how accurate it is computed in the smaller model space matrix.*

as Fig. 5.16 indicates, that even with a matrix dimension of 160 000, the eigenvalues are far from converging. There is still too much information in the higher energy levels that has not been taken into account so far. The eigenvectors reflect this as a eigenvector of a smaller model space will not reach a higher precision than $< 10^0$ when projected onto the higher model space, see Fig. 5.17. The figure also shows that there is no need to compute the eigenpairs to the smaller modelspace with a precision lower than 10^0.

The particular matrix studied was for a system of six particles. Perhaps the convergence between the model spaces are faster for systems of fewer, but in this case far larger matrices need to be studied before this starting vector approximation might become more useful. Anyhow, a start approximation of tolerance 10^0 is not completely wasted. According to Fig. 5.11 the iterations might be reduced with approximately 20 by using the approximation in that context, which could be worthwhile since the smaller model space matrix is considerably smaller. To compute the eigenpairs with a precision of 10^0 from that start approximation will not take many iterations either. However, a high precision on eigenpairs of matrices to model spaces that changes this much between two successive spaces is rather uninteresting. There is not much point in computing an eigenvalue to a precision that is higher than the accuracy of the matrix, as an approximation to the true physical system.

It is possible that the use of approximations from smaller model spaces are of greater use as the matrices grows really large and the eigenpairs changes less between the spaces.

5.5 A short survey of available software

Since an actual implementation of an eigensolver (for purposes other than quick evaluations of different methods) was not in the scope of project a short survey of available software has been conducted. The findings are summarized in table 5.7 where a particular routine or library is specified by eight columns. The first five contains the name, type of recursion, reorthogonalization technique and the programming language used. The sixth column indicates whether the user is required to supply a separate routine for calculating the matrix-vector product Av, abbreviated by USMP (User Supplied Matrix Product). Similarily, the seventh column indicates whether the user is free to control the computation of any other vector or matrix products necessary on request by the program (User Supplied Vector Products) or if these are handled internally by the driver. The final column shows if the software can be run in parallell either using a message-passing paradigm via MPI or a shared-memory paradigm via OpenMP.

Note that this table exclusively lists eigensolvers based on Lanczos methods, hence eigensolvers based on e.g. Davidson-methods and software using Lanczos methods for purposes other than finding eigenvalues (i.e. solving linear equations or singular value decomposition) have all been excluded. Following the table is a short description of each entry, links to repositories containing source code and documentation can be found in Appendix D.

Table 5.7. *Available software for solving LSSEs.*

Name	**Recursion**	**Restart**	**Orthogonalization**
IRBLEIG	Block	Implicit	Partial
ARPACK	Vector	Implicit	Full
BLZPACK	Block	Explicit	Partial/Selective
LASO	Block	Explicit	Selective
LANCZOS	Vector	None	None

	Language	**USMP**	**USVP**	**Parallell**
IRBLEIGS	Matlab	Yes	No	No
ARPACK	FORTRAN 77	Yes	No	MPI
BLZPACK	FORTRAN 77	Yes	No	MPI
LASO	FORTRAN IV	Yes	No	No
LANCZOS	FORTRAN	Yes	No	No

IRBLEIGS is based on the Block Lanczos algorithm and currently only available in MATLAB. IRBLEIGS combines several advanced features such as an implicit restarting scheme to accelerate convergence, partial reorthogonalization and shift-invert strategies for calculating interior eigenvalues.

ARPACK[5] is a library of subroutines written in FORTRAN 77/C++ for solving large eigenproblems. The implementation is centered around the Arnoldi algorithm which reduces to the Lanczos algorithm in the case when A is symmetric. A full reorthogonalization is employed to guarantee working-precision orthogonality among the Lanczos vectors and furthermore, ARPACK uses an implicit restarting scheme (briefly discussed in Section 4.3).

BLZPACK is a Block Lanczos-solver written in FORTRAN. BLZPACK mainly uses modified partial reorthogonalization to maintain semi-orthogonality between the Lanczos blocks. In the event that the process runs out of memory, BLZPACK performs an explicit restart while storing any converged Ritz vectors and orthogonalizing with respect to these vectors in order to avoid having to recompute any sufficiently converged eigenvalues (this ismore appropriately referred to as external selective orthogonalization).

LASO is another Block Lanczos-solver written in FORTRAN IV and notably the only one using selective reorthogonalization as described in Section 4.4.5. Similarly to BLZPACK, it uses an explicit restart in the event that it runs out of memory.

LANCZOS is a FORTRAN implementation of a vector Lanzos method using no reorthogonalization. Spurious eigenvalues are systematically detected and removed by a scheme for post-processing the tridiagonal matrix T, see Section 4.4.1.

Of the available solvers included in table 5.7, only ARPACK and BLZPACK can be run in parallel (The parallel implementation of ARPACK is called P_ARPACK), a necessary feature in order to tackle any truly large-scale computations. While all of the solvers listed above allowed for a user specified matrix-vector product, seemingly none of them allowed for user specified vectors products. However, this was not always clear from the documentation and for the cases where no explicit mention of this feature was made or could be easily deduced from the code the answer was assumed to be negative.

[5] ARPACK is also available in MATLAB through the function `eigs`.

6 Analysis and adaptations of the NCSMb code

In the two previous chapters the eigensolvers Lanczos and Block Lanczos have been dealt with. In Chapter 4 the underlying mathematical theory was thoroughly explained and presented together with explicit outlines of the algorithms, taking into account the effects of finite precision etc. The succeeding Chapter 5 presented various tests made with the Lanczos algorithm investigating different aspects such as the width of the starting block, reorthogonalization schemes and how accelerate convergence.

The following two chapters will instead put focus on the code No-Core Shell Model for bosons, NCSMb, and the properties of the matrices it calculates. In Section 6.1 analysis of NCSMb is presented, where certain areas of improvement have been identified. Based on these areas, adaptations have been made to the code, which are discussed in Section 6.3 and Section 6.5.

Understanding the reasoning behind the adaptations made to the code however requires knowledge of some formats and methods used. In Section 6.2 a storage format is introduced, used to effectively store large, sparse and symmetric matrices, while in Section 6.4 the main features are explained of a hash table; a structure that provides a quick lookup of values given that the hash table is well-dimensioned and the hashing function is chosen with care.

6.1 Analysis of the code NCSMb

The code NCSMb has several arguments: the cut-off level of the energy of the model space $N_{\max}$, the number of particles N_{boson}, the projection of the total angular momentum M_{total} and the choice of two- and/or three body operators, with energies E_2 and E_3.

Using these arguments it calculates the matrix elements of the resulting matrix representing the Hamiltonian using a for-loop over all rows of the matrix. A hash table is used to find the connections of the many-particle states that generate additions to the matrix elements. The connections in turn are found by taking a many-particle state and for each such state, iterate over all possible many-particle states that can generate nonzero matrix

elements, when computing the interaction of two- and/or three-body operators. Due to the way the elements are calculated, the updates to the elements in a row in the matrix are not made in any particular order. Several updates can also be made to each elements as there can be several connections between many-particle states. A somewhat more detailed account is given in Section 3.4. The matrix elements are thereafter stored in the format of `gsl_matrix`[1], which is done to be able to compute the eigenvalues of the matrix using `gsl_eigen`[2].

While analyzing the code NCSMb the following areas of improvement were discovered:

– **final storage of the matrix elements**
  As the Hamiltonian matrix is large and sparse, storage space would be much less by only storing the non-zero elements, together with their positions in the matrix.

– **storage of the matrix elements during computations**
  Depending on the number of updates made to each matrix element during calculations of each row, a *hash table* could be advantageous at this stage, instead of a large array for storing both the zero and nonzero elements.

6.2 Storing symmetric sparse matrices

To store a sparse Hermitian matrix as efficiently as possible the properties of the matrix; the sparseness and the symmetry, should be used in order to reduce the number of matrix elements that needs storing. Due to the sparseness, only the relatively few non-zero elements must be stored together with their position in the matrix. The symmetry also allows storage of only the upper- or lower triangular half of the matrix together with the diagonal.

When it comes to storing the non-zero elements there are two choices: go through each column and store the elements as they appear in the matrix (column-major format), or go through each row (row-major format). One of several ways of storing the matrix is the *compressed sparse row format*, (CSR), [21] specified by the following three arrays:

`values:` an array containing the nonzero elements of the matrix, stored when walking along each row.

[1] a matrix storage format in the GNU Scientific Library (GSL).

[2] a numerical routine in the GNU Scientific Library (GSL) for calculating the eigenvalues of a matrix stored in the `gsl_matrix` format.

`columns`: an array in which element i contains the column number of element i in `values`.

`rowIndex`: an array in which element j contains the index of the element in `values` which corresponds to the first nonzero element of row j.

Should row j contain no nonzero elements, the value of the next element in `rowIndex` will be the same, i.e.`rowIndex(j+1) = rowIndex(j)`. This means that the number of nonzero elements in row j can be found through the difference `rowIndex(j+1)` − `rowIndex(j)`. As this must hold for all rows, an additional entry is added to the end of `rowIndex`, where its value is the total number of nonzero elements plus one or zero, depending on if the last row contains a nonzero element or not.

The arrays `values` and `columns` will have as many elements as there are nonzero elements in the matrix, while `rowIndex` will have $d + 1$ elements, where d is the dimension of the matrix.

Example. (*CSR storage*) A symmetric matrix A is to be stored in the CSR format, and the matrix elements will be stored from the upper diagonal. The matrix A has the following appearance

$$A = \begin{bmatrix} 1 & 0 & -1 & 3 \\ 0 & 2 & 4 & 0 \\ -1 & 4 & 0 & -2 \\ 3 & 0 & -2 & 5 \end{bmatrix} \tag{6.1}$$

The three arrays specifying the matrix, with one-based indexing, will then become:

values	1	-1	3	2	4	-2	5
columns	1	3	4	2	3	4	4
rowIndex	1	4	6	7	8		

Table 6.1. *The three arrays defining the matrix A, according to the CSR format.*

■

6.3 Implementing CSR format storage in NCSMb

Since the code NCSMb stored the entire matrix, nonzero and zero elements, in the format `gsl_matrix`, storage space could be saved by implementing a sparse storage format. As a future implementation of an eigensolver will be based on the Lanczos algorithm, many matrix-vector multiplications will be performed, and a sparse format is advantageous in making the computations effective. The format chosen for output from the NCSMb code was CSR, as described in Section 6.2.

Due to the symmetry of the matrix, only the matrix elements of the upper *or* lower part needed to be stored. The upper part was chosen as the part to store the elements from, meaning only the elements of this part had to be calculated.

As a first step the array `oneRow` was introduced, with length of a row in the matrix, which stored all the elements in a row, zero or nonzero. This was repeated in each iteration of the loop over the rows. After each row a for-loop went through `oneRow`, storing the nonzero elements into `values` and their position in the matrix into `columns`, as described in Section 6.2. The next entry of `rowIndex` was filled to indicate the number of nonzero elements in the row.

6.4 Hash tables

A hash table is a data structure that allows one to map certain identifying *keys*, to their associated *values* and store them in an array [22]. The mapping is realized using a hash function that accepts a key, transforms it in an appropriate way, and returns a *hash coding*. The hash coding will correspond to an index of an array where the associated values can be found, which means a value can be put into the hash table using a key together with the hash function, and later it can be retrieved with the same procedure. A schematic diagram of a simple hash table with its hash function can be seen in Fig. 6.1, where the full names of John, Lisa and Sandra act as keys while their phone numbers act as values.

The main advantage of using a hash table as means of storage and lookup over other data structures is due to the speed. The lookups[3] will in principle be constant in time up until a occupancy level of 2/3 of the total size of the table. If the number of entries in the hash table can be predicted beforehand, this reduces the time taken for lookups as no resizing of the hash table is needed.

[3] A lookup is the process of searching through a data structure for a specific value.

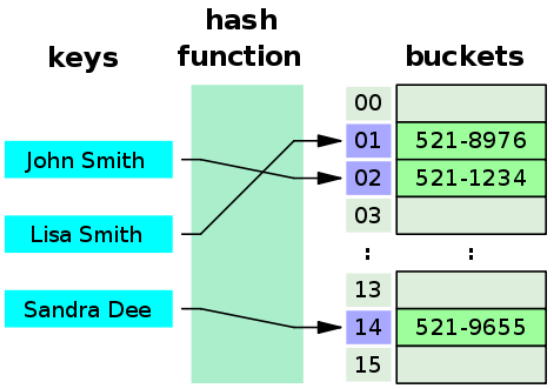


Figure 6.1. *A small phone book as hash table, where the* buckets *indicate the arrays where values are stored in the hash table.*[1]

6.4.1 Choosing hash function

A perfect hash table would use a hash function that uniquely mapped all keys to indices of the array, thereby avoiding any problems associated with two keys mapping to the same position in the table; a so-called *hash collision*. In practice however, this is rarely achievable meaning collisions will occur, and must be dealt with in appropriate ways. Still, the number of collisions can be kept to a minimum by choosing a good hash function; one that distributes the generated indices evenly over the array, and minimizes micro-clustering of the hash values.

6.4.2 Handling collisions

When it comes to collisions, two different ways of dealing with them are *chaining* and *rehashing*, both with their advantages and drawbacks.

A chained hash table will have linked lists at every position in the array, called *buckets*. An example can be seen in Fig. 6.1. When inserting a new element into the table, the hash function returns an index for the element, and the value together with the key will be put in the bucket, at the beginning of the linked list of that position. If another element receives the same index, that key and value will be placed at the next available position in the list, and the collision has thereby been dealt with. However, with many element stored in the same bucket the linked list will be long, and a search

[1]Reference: `http://en.wikipedia.org/wiki/File:Hash_table_3_1_1_0_1_0_0_SP.svg` by Jorge Stolfi. Licensed under the Creative Commons Share-a-Like-license: `http://creativecommons.org/licenses/by-sa/3.0/deed.en`

through the list will take its time, negating the high efficiency of finding the bucket in constant time.

In a hash table that uses rehashing, or resizing of the hash table, all the elements are stored in the actual array. When an element is to be stored in a position where an other element already is placed, the table will be probed for other available positions. A linear probing of a hash table, where the initial index of the value is $h'(k)$ and the index after i iterations is $h(k,i)$, can be expressed as follows:

$$h(k,i) = (h'(k) + i) \bmod m, \tag{6.2}$$

with the iteration variable $i = 0,1,\ldots,m-1$ and with m positions in the hash table. To find an available position the index is increased by i, and the modulus is taken of the length of the hash table, to ensure that the new index does not exceed the maximum index of the hash table. This procedure is repeated until an available position is found.

A condition could also be set concerning the maximum number of hash collisions that are allowed for a single hash entry, as a considerable amount of hash collisions will require much longer time for the hash table to be filled.

6.4.3 Dynamic resizing

In order to make the storage as useful as possible, the hash table can be made to change its size to adapt to the number of entries in the table. This is practical since the number of entries might not be known, but having a very large hash table will waste time and storage. At the same time a too small hash table will cause lots of collisions, and this should be avoided as dealing with them takes time. Using a rehashing function the size of the table can be adapted when the number of entries reaches a critical value. At this point, the size of the hash table is increased to a larger size and the rehashing function converts the old indices of the hash table into new indices, as the length of the table now has changed. The keys and values are stored in their new positions in the new hash table.

6.5 Implementation of a hash table in NCSMb

Chapter 7 presents results from a number of investigations performed to find out the matrix properties, where Fig. 7.5 shows the average number of updates of the elements in the matrix against dimension. From the curves in the figure and in the discussion of Section 7.2.1 the implementation of a hash table is motivated, since the average number of updates is low for both two- and threebody-forces and the number of elements will thus be few compared to the dimension of the matrix.

The implemented hash table handles collisions by rehashing, and uses dynamic resizing. The variable length was motivated both by the possibility to handle many collisions, but also due to that only the upper diagonal of matrix needs to be calculated, and the number of available elements will decrease as the loop iterates through the rows. The size of the hash table will be notably smaller than the dimension of the matrix and contain much fewer elements to go through.

The implementation is done in the for-loop over the rows. Instead of the use of an array `oneRow`, which stored all the matrix elements and then had to been iterated through to collect the nonzero elements, a hash table with allocated memory of one row is introduced. The actual length of the hash table however, is set at the beginning of the loop, to a length adapted according to the number of nonzero elements of the previous row. In the first loop the length of the hash table is set to some predetermined value.

If one row should contain unexpectedly many nonzero elements, a function `reHash` will adapt the size of the hash table according to a new standard, and translate the old indices into new ones. The size of the new hash table will however still be small compared to the dimension of the matrix.

The hash function of the table is written so that the value of the column of the matrix becomes the key, taking the modulus of the current size of the hash table. This actually provides a quite uniform distribution of the indices as the updates made to elements in the matrix are rather well spread out over the columns, except for several updates on the diagonal. The hash function is therefore $index = key \bmod l$ where l is the current length of the table.

7 Investigations of matrix properties

In order to find further possibilities and limitations of an eigensolver based on the Lanczos algorithm, the structure and traits of the matrices to be diagonalized were investigated. Depending on for example the sparseness of the matrices or how quickly the dimension grows with the cut-off energy $N_{\max}$, decisions can be made on how large matrices it is reasonable to diagonalize or how much storage space that will be needed. This chapter will therefore present the results of several tests made with the NCSMb code to find details on the number of nonzero elements, dimensions related to the cut-off energy, and number of updates per element, among others. Note that every calculation that used three-body forces also used two-body interactions, but not vice versa. In NCSMb one could specify the two- and three-body potentials with the energies E_2 and E_3 respectively, and both of them were set to one in all tests. The actual values only for very particular choices affect the structure of the matrix.

7.1 Storage size of the matrix

Knowledge of the size of the matrix and the number of nonzeros for a given dimension helps in predicting the memory usage for large matrices. This information is important to be able to prepare appropriate hardware and memory for large-scale computations of the eigenenergies. It was therefore investigated how the dimension of the Hamiltonian increases for growing model spaces, and how the number of nonzero elements depend on dimensions.

7.1.1 Relation between matrix dimension and $N_{\max}$

In order to know the storage size we first investigated the matrix dimensions for various values of cut-off energy $N_{\max}$ and the number of particles N_{boson}. This was done for relatively small dimensions as computing very large matrices consumes a lot of time and memory. Nevertheless, reasonable estimates of the matrix dimension can be made for larger model spaces using the obtained data.

We used the NCSMb code to compute the dimensions of matrices for $N_{\max} =$ 2–22 and $N_{\text{boson}} = 2, 3, 4, 5, 7, 10, 15, 20$ and 30. The results are plotted in Fig. 7.1, showing both a normal and a logarithmic scaled graph. It appears that the dimension is independent of the number of bosons when N_{boson} reaches a specific value, a value that does in fact depend on $N_{\max}$. It can be found numerically that the dimension becomes independent of N_{boson} approximately when $2N_{\text{bosons}} > N_{\max}$. Hence, if this condition holds, the dimension will not be influenced further when increasing the number of bosons while holding the value of $N_{\max}$ constant.

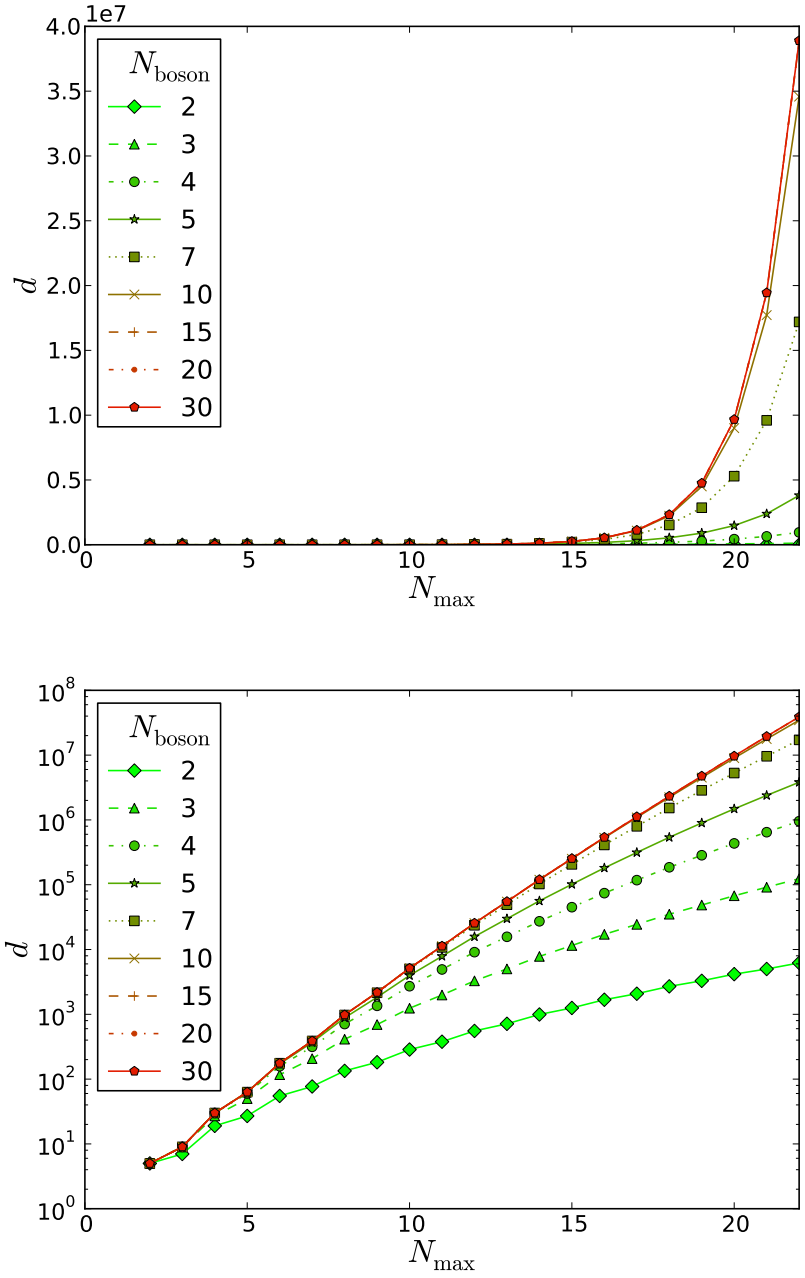


Figure 7.1. *The dimension d of the Hamiltonian matrix in normal (above) and logarithmic (below) scale, for N_{max} = 2–22 and various numbers of particles N_{boson}.*

7.1.2 Number of nonzero elements

Calculating the relationship between the number of nonzero elements N_{nz} and dimension is the next step in understanding the memory consumption of large model spaces. The number of nonzero elements in the matrix is expected to grow as $N_{\text{nz}} = d^x$, with the exponent x having a value of $1 \leq x \leq 2$. A value of $x = 1$ implies that only the diagonal of the matrix contains nonzero elements, while $x = 2$ instead implies that the entire matrix is nonzero.

To find the exponent x the number of nonzero elements was calculated for various N_{max} and N_{boson} and stored together with the dimension of the matrix. Furthermore, as the number of nonzeros depends on whether three-body forces are incorporated or not, the same tests were made with and without three-body interaction. The results of these tests can be seen in Fig. 7.2, where it is observed that there seems to be a trend indicating that the number of nonzero elements indeed grows as $N_{\text{nz}} = d^x$. Since it was observed in Section 7.1.1 that the dimension mainly grows with N_{max}, and is only influenced by N_{boson} up until a certain value, curves with different values of N_{boson} are plotted to display the results most effectively. In addition, Fig. 7.3 shows the fraction $r_{\text{nz}} = N_{\text{nz}}/d^2$ of nonzero elements with respect to the dimension of the matrix.

With $N_{\text{nz}} = d^x$, it follows that $\log_{10}(N_{\text{nz}}) = x \cdot \log_{10}(d)$. The value of x can thereby be found as the gradient of a plot of $\log_{10}(N_{\text{nz}})$ against $\log_{10}(d)$, as seen in Fig. 7.4. In the figure it can be observed that as the number of bosons N_{boson} increases the gradient of the straight lines for the different values of N_{boson} converges approximately towards a limiting line, for both the case of two- and three-body forces.

A linear fit was performed with the curves of $N_{\text{boson}} = 30$ in both cases, and the results are shown in Fig. 7.4 (a) and (b). For the case of two-body forces this results in the linear equation

$$\log_{10}(N_{\text{nz}}) = 1.44 \log_{10}(d) + 0.52, \tag{7.1}$$

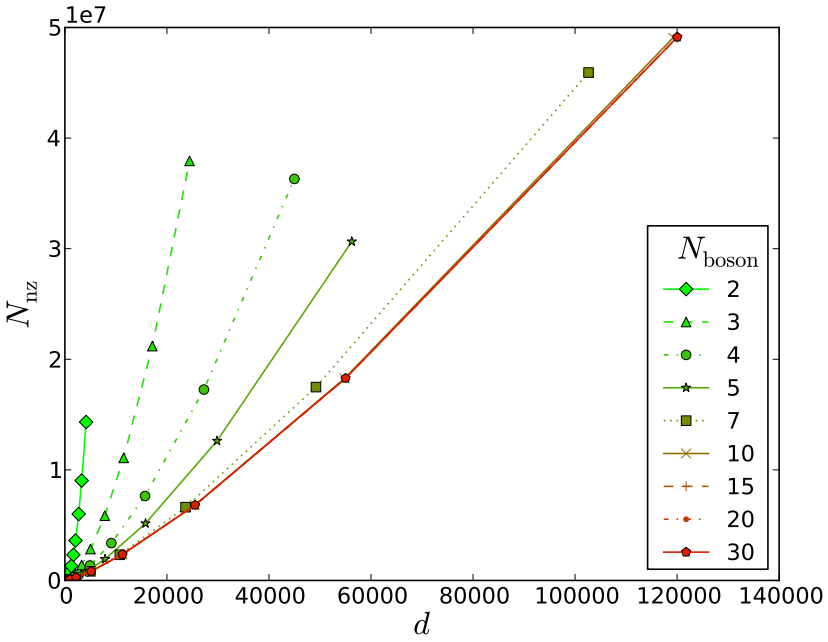
while in the case of three-body forces it results in the following equation:

$$\log_{10}(N_{\text{nz}}) = 1.75 \log_{10}(d) + 0.26. \tag{7.2}$$

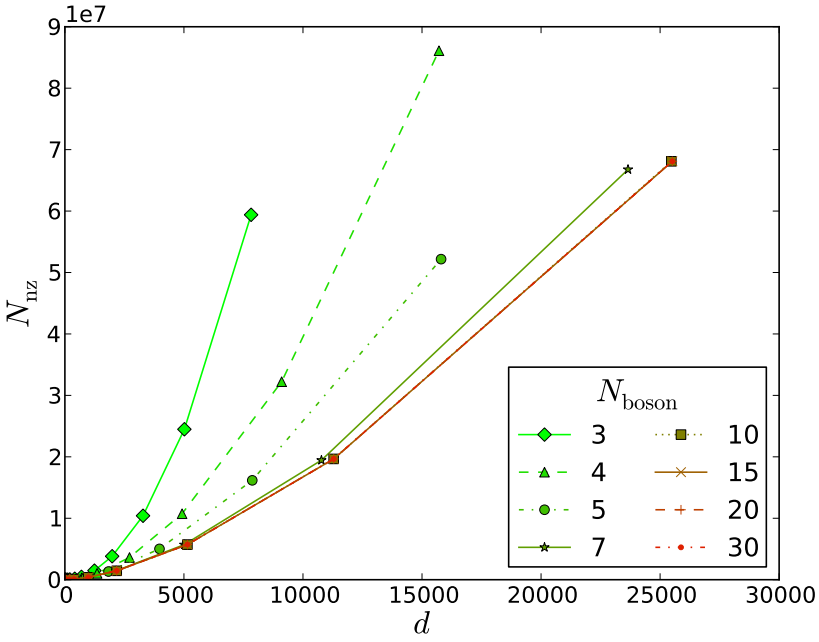
This means that for the two different types of forces, the nonzero elements grow approximately as

$$N_{\text{nz},2} = d^{3/2}, \tag{7.3}$$

$$N_{\text{nz},3} = d^{7/4}. \tag{7.4}$$

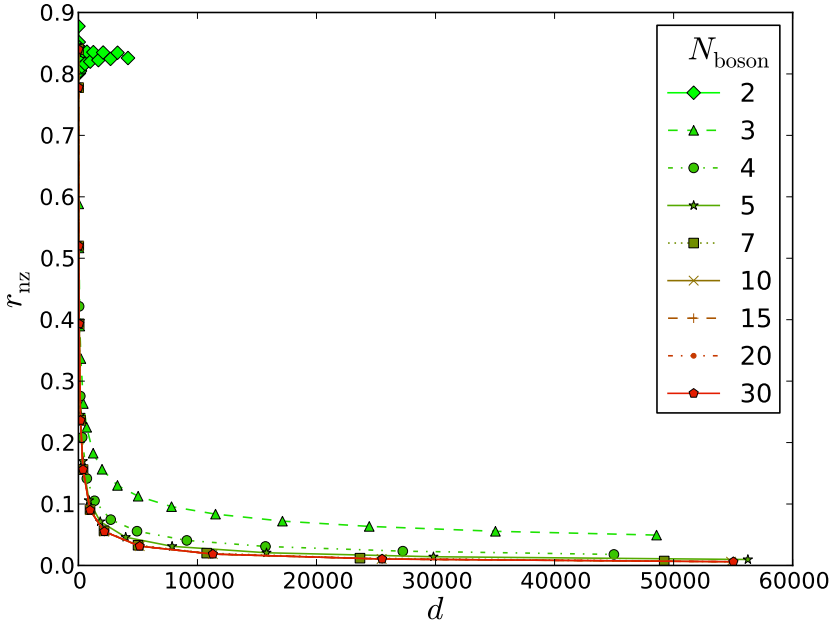


(a) Two-body forces.

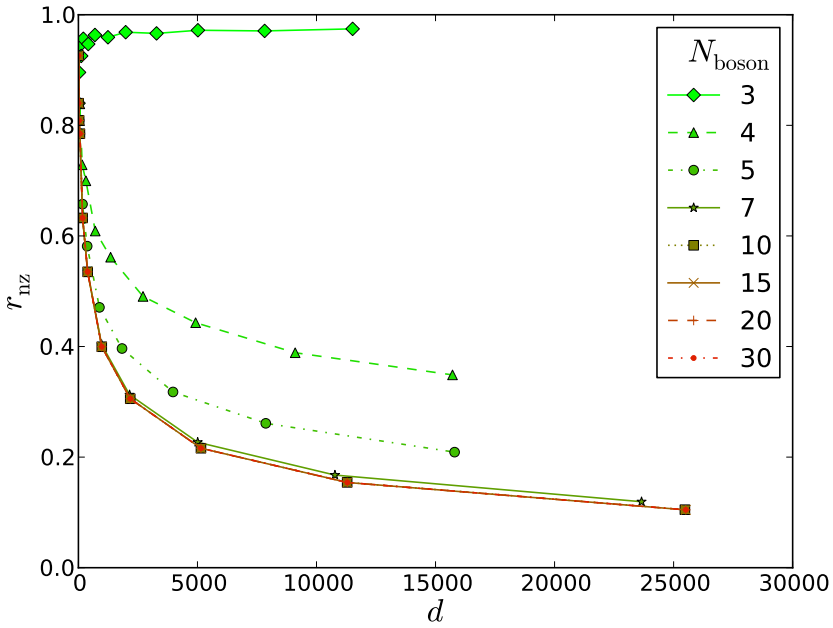


(b) Two- plus three-body forces.

Figure 7.2. *The relationship between the number of nonzero elements* N_{nz} *and the dimension* d*, for various values of* N_{max} *and* N_{boson}.

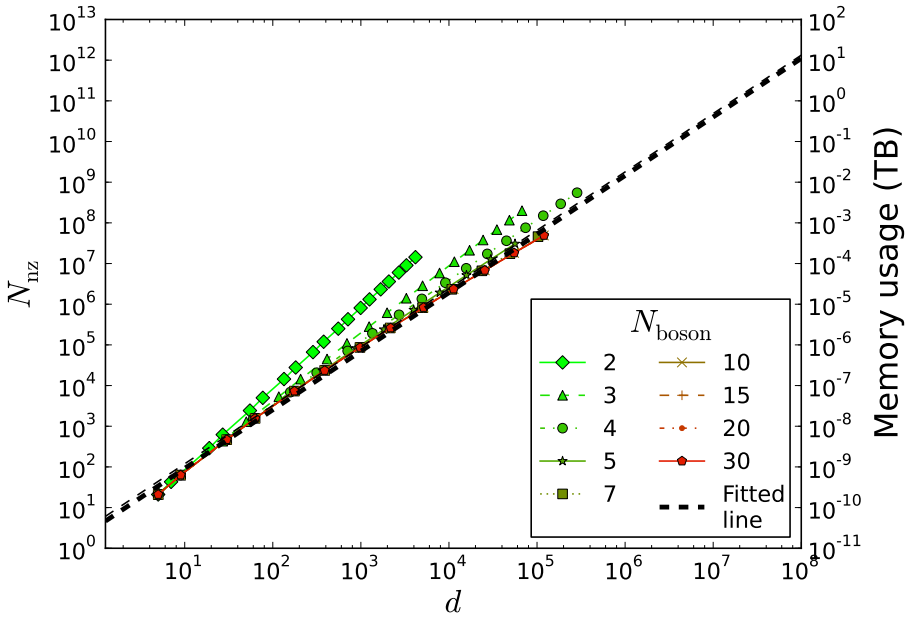


(a) Two-body forces

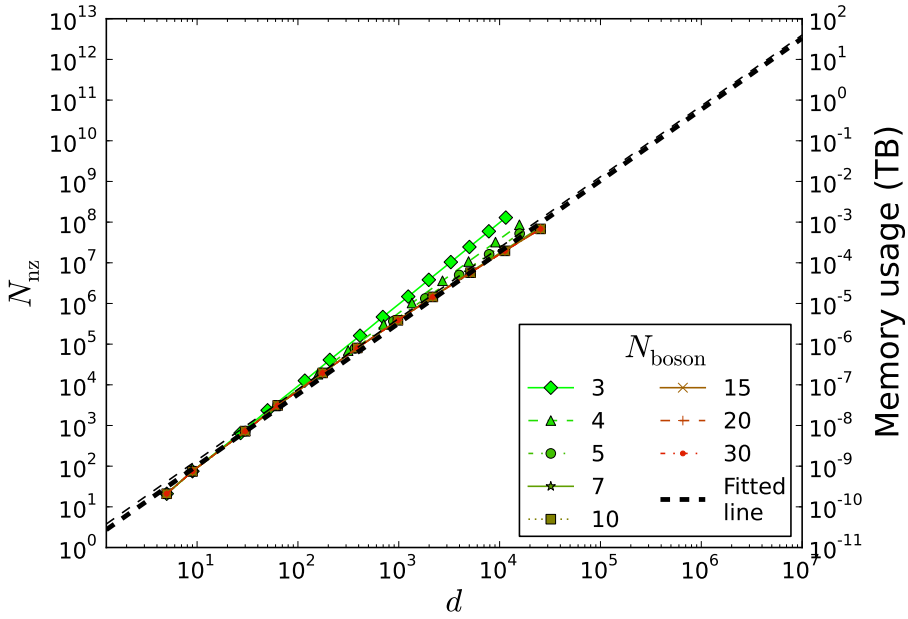


(b) Two- plus three-body forces

Figure 7.3. *The relationship between the fraction r_{nz} of nonzero elements and the dimension d.*



(a) Two-body forces.



(b) Two- plus three-body forces.

Figure 7.4. *The number of nonzero elements and memory usage, in terabytes (TB), against dimension d.*

7.1.3 Storage size of matrix against dimension

After computing the number of nonzero elements and the dependence of the dimension on $N_{\max}$ and N_{boson} a prediction can be made of the storage size of the Hamiltonian matrix. The storage of the matrix requires, based on the CSR format explained in Section 6.2, storing the three arrays `values`, `columns` and `rowIndex`. As double precision is a usual choice of format for the matrix values, each nonzero value would need 8 bytes of memory, and every column and row index is an integer which consumes 4 bytes[1]. Therefore, the total storage size of a matrix is given by

$$\Big[(8+4)\cdot N_{\text{nz}} + 4\cdot(d+1)\Big] \text{ bytes} \approx \Big[12N_{\text{nz}} + 4d\Big] \text{ bytes}, \tag{7.5}$$

since the number of elements in `rowIndex` is $d+1$, and there are N_{nz} elements in both `values` and `columns`. Thus, all that is needed are N_{nz} and d, which we already have, and by extending the linear fit found previously we get an idea of how much memory is needed for matrix calculations of much larger scale. The memory usage is greatly dependent on the inclusion of three-body potentials as N_{nz} is quite different between the two cases. Therefore a second scale can be seen in Fig. 7.4, showing the total storage size as a function of d. In the figures the memory usage has been scaled in terms of terabytes (TB). To see the difference between two- and three-body forces the storage space of 1 TB could be taken as example, which allows storage of a matrix of dimension $d = 10^7$ in the case of two-body forces, while this value is $d = 10^6$ for three-body forces.

7.2 Properties of the matrix elements

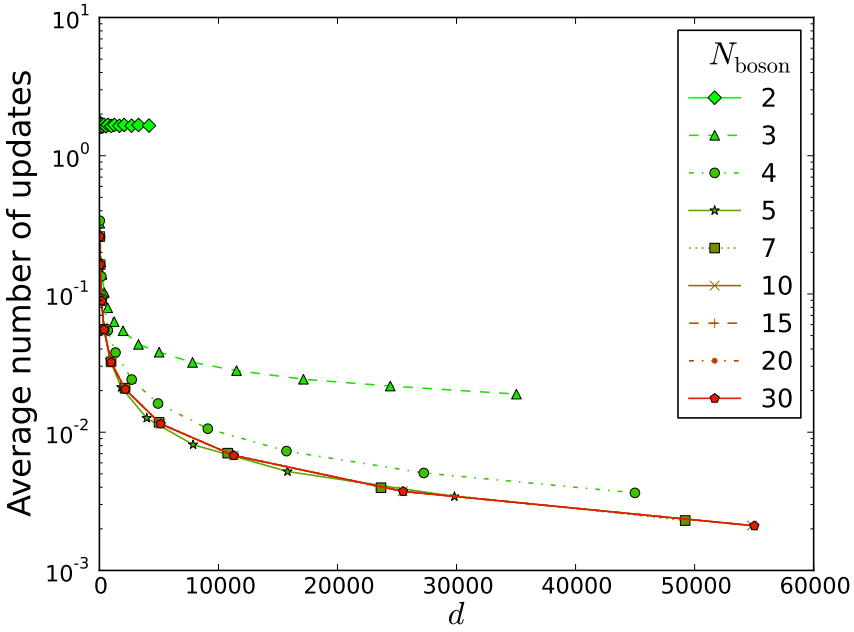
In our investigations of the Hamiltonian matrix some properties were found of how the NCSMb code generates the matrix elements and how the elements are distributed. This kind of investigations are helpful to e.g. optimize the matrix generating code as well as determine how to split the matrix to different CPUs, when performing calculations in a future eigensolver. For instance, knowing the average number of updates of the elements motivated us to implement a hash table that reduced the execution time of the algorithm.

[1] If the dimension is large enough, 8 bytes may be needed for each index.

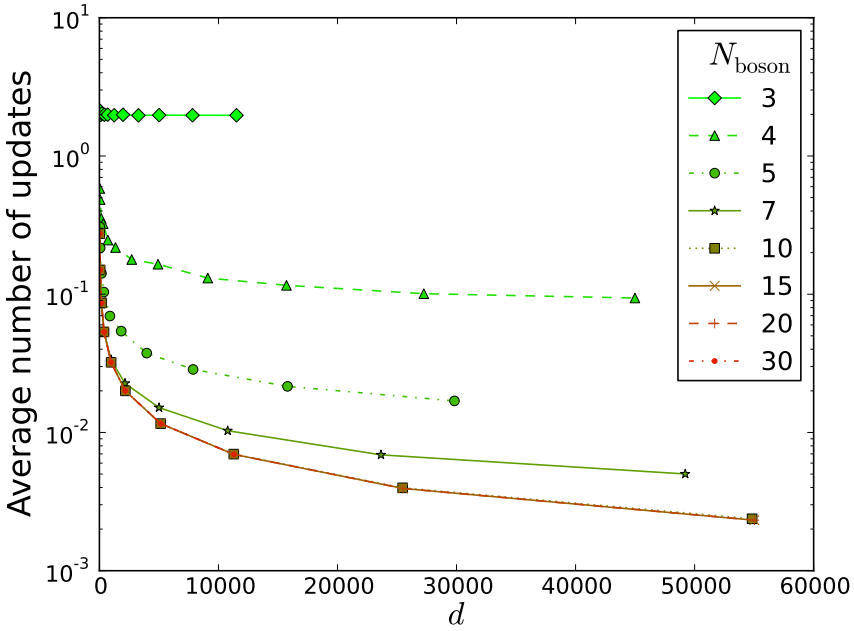
7.2.1 Number of updates per element

The use of a hash table for storing the nonzero elements of one row would only be valuable if the number of updates per element was reasonably low compared to the dimension of the matrix, i.e. the length of one full row. With many updates per matrix element, many lookups in a hash table would be required, and the cost of lookups and handling collisions might not be lower than the cost of searching for nonzero values in an array the size of one row.

The average number of updates per matrix element have been calculated from the code NCSMb, and this is shown in logarithmic scale against dimension in Fig. 7.5, for both the case of two- and three-body forces. The average is taken over the entire matrix, including both zero and nonzero elements, and the number of bosons varies according to $N_{\text{boson}} = 2$–30. As can be seen in the figure the average number of updates decreases rapidly as the dimension grows, and this is the case for all values of N_{boson}. At a value of $d = 10^4$ the average number of updates is below 10^{-2} for two-body forces and below 10^{-1} for three-body forces, when $N_{\text{boson}} \geq 4$. It can also be noted that except for the smallest value of N_{boson}, the average number of updates is well below 1.



(a) Two-body forces



(b) Two- plus three-body forces

Figure 7.5. *The relation between average number of updates per matrix element and matrix dimension d.*

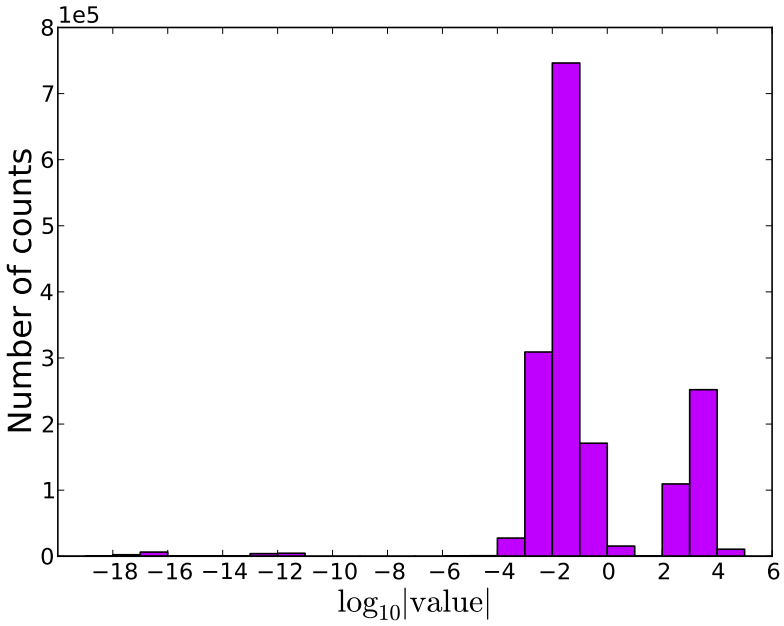
7.2.2 Values of the matrix elements

Fig. 7.6 shows the distribution of the absolute values of the matrix elements, for both the two- and three-body case. The number of counts is one order of magnitude larger for three- than two-body forces, as expected since the matrix resulting from three-body interactions is less sparse. We note that a significant number of elements have values less than 10^{-12}, and the guess is that these elements emerge due to numerical errors and should actually correspond to elements that are summed up to zero. This is not an unreasonable guess since the next chunk of nonzero values are several orders of magnitude greater. It can also be noted in Fig. 7.6 that there are two groups of elements situated around 10^{-2} and 10^{3}, even though the latter is more prominent in the case of two-body forces.

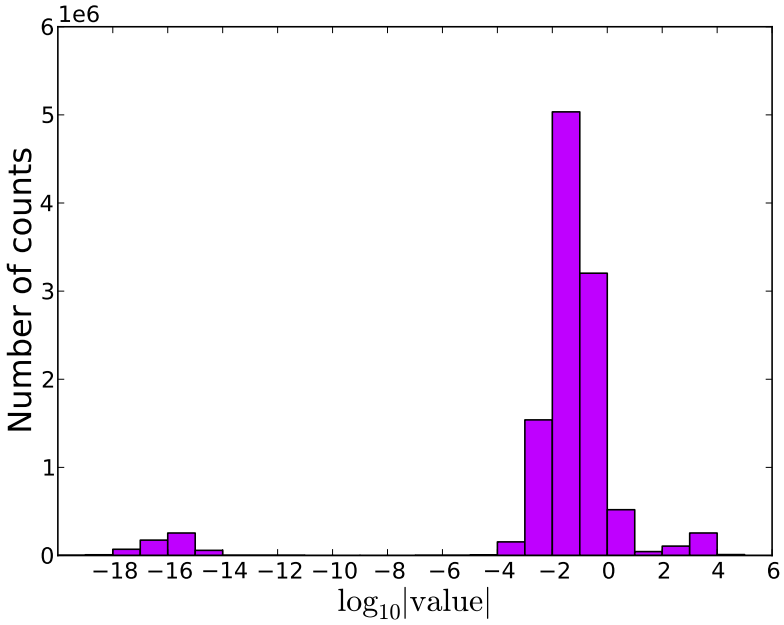
7.2.3 Distribution of nonzero elements over the matrix

In order for the matrix-vector multiplications of the eigensolver to be performed as effectively as possible, it is interesting to analyze how the nonzero elements are distributed over the matrix. A future implementation of an eigensolver will most likely use parallel threads to perform calculations, and depending on the distribution of the nonzero elements the matrix can be split into blocks with equal or different sizes. In Fig. 7.7 the distribution of the nonzero elements over the matrix can be seen for both the case of two- and three-body potentials, where dark areas indicate a concentration of nonzero elements. The middle pictures show the matrix with decreased resolution to observe the patterns more clearly.

In order to visualize where the largest elements are situated in the matrix, (arbitrary defined as those with absolute values larger than 10^2), 2D plots of the matrix with only large values are shown in the bottom panels of Fig. 7.7, for the cases $N_{\text{boson}} = 10$ and $N_{\text{max}} = 10$. Note that both the numbers and the locations of the largest absolute values are almost the same.



(a) Two-body forces.



(b) Two- plus three-body forces

Figure 7.6. *The number of elements of a given value, for a matrix with dimension* $d = 5147$. *The parameters were* $N_{\max} = 10$ *and* $N_{\text{boson}} = 10$.

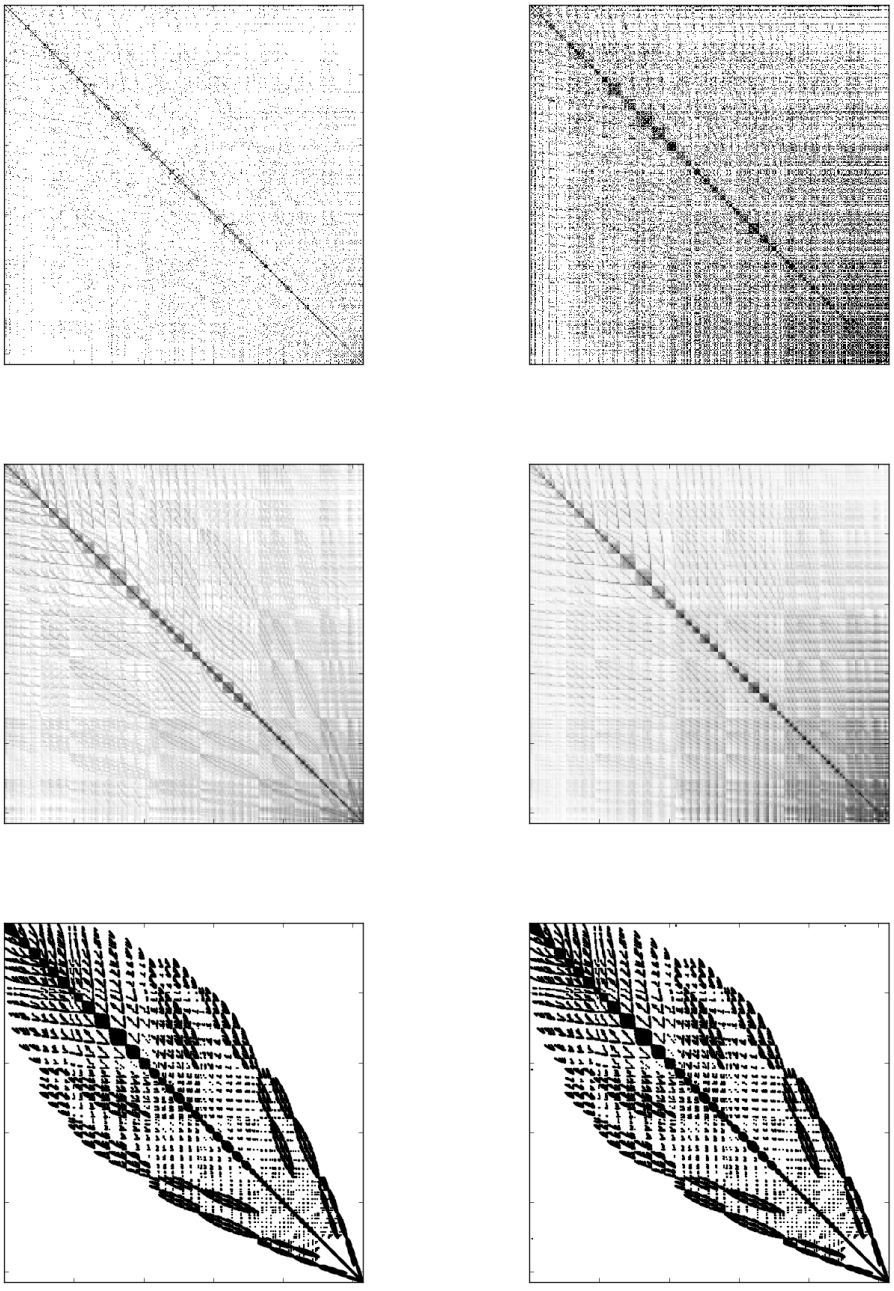


Figure 7.7. *The distribution of nonzero elements of the Hamiltonian for* $N_{\mathrm{boson}} = 10$*,* $N_{\mathrm{max}} = 10$*, and* $d = 5147$*. The left panels are with only two-body interactions, and the right panels include three-body forces. The center panels show the matrices with lower resolution, for clarity. The bottom panels show the distribution of the elements whose absolute value is larger than* 10^2*.*

7.3 The number of single-particle states

If a many-particle system has the cut-off energy $N_{\max}$ there is a number of single-particle states that in principle can be constructed. However, some of these are not available to the many-body system because of the restriction that all many-body states must have the same total M (for details see Section 3.4). By not generating the unused states the indexing of the states becomes dense, reducing the storage needs for auxiliary arrays using these indices.

In Fig. 7.8 a comparison can be seen between the number of originally enumerated single-particle states in NCSMb and the ones actually used, for different $N_{\max}$. The number of single-particle states have no dependence on N_{boson}.

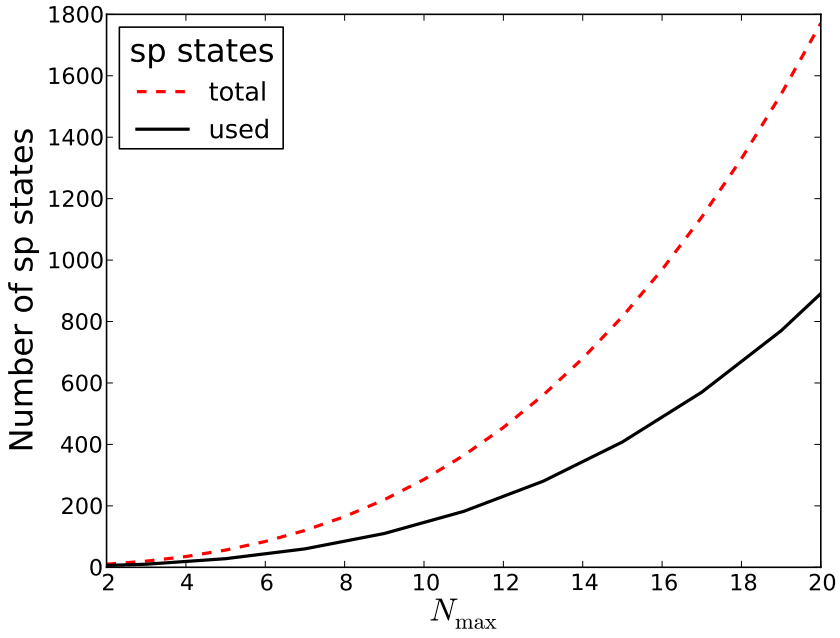


Figure 7.8. *The number of single-particle (sp) states with respect to* $N_{\max}$.

8 Discussion

8.1 Lanczos and Block Lanczos methods

In Chapter 5 a thorough investigation of different aspects of the Lanczos and Block Lanczos method was presented. The aim of this section is to provide a short summary of the results obtained from these investigations as well as a discussion of the possible implications for a future implementation of an eigensolver in the NCSMb code.

8.1.1 Choosing a block size

In Section 5.2 it was shown that an increase in block size reduced the number of multiplications with the matrix A required for convergence. For a truly large-scale eigenproblem, when the dimension A may be of the order $\gtrsim 10^9$, the cost of computing matrix-vector products involving A is believed to dominate the cost of the entire Lanczos run, thus providing preliminary evidence in favor of operating with blocks instead of a single vector. The situation is somewhat complicated by the fact that multiplying A with a set of vectors rather than a single vector comes at a price. It turns out that some block sizes are more preferable than others in this regard, as they manage to fill the cache-lines more effectively, see Section 5.2. To get a realistic estimate of the effective computational time needed for convergence, numbers based on a preliminary investigation using a cache-friendly routine for sparse matrix multiplication were used to scale the number of multiplications with A used by the Block Lanczos. In the end, the results favored a Block approach when several eigenpairs were required. In particular, if the number of eigenvalues sought is less than ten, the optimal choice seems to be a block of size $p = 4$. If more eigenvalues are required, a size of $p = 8$ seems to work equally well.

8.1.2 Choosing a reorthogonalization technique

Section 5.3 investigated the various schemes for restoring orthogonality among Lanczos blocks (or vectors) which were discussed in Section 4.4. The main contestants were full reorthogonalization (FRO) and partial reorthogonalization (PRO), of which FRO is the more conservative choice. The comparison was based on two factors: the actual number of orthogonalizations

performed and the number of recall operations where previously generated Lanczos blocks have to be scanned and retrieved from primary memory or possibly fast secondary storage. On both counts, PRO was found to be the superior choice, using only a third of the number of recalls used by PRO and cutting the required number of orthogonalizations in half for reasonable choices of the block size. Data indicating (see Figs. 5.7 and 5.9) how these numbers scaled with block size and dimension of the Krylov space was also provided for future estimates of the cost of using PRO for a given problem. This data showed that PRO gives diminishing returns as the block size increases beyond $p = 10$, thus lending further support for a restrictive choice of the block size.

The second part of Section 5.3 was comprised of a comparison between the PRO scheme and a closely related scheme called modified partial reorthogonalization (MPRO). The motivation behind using MPRO rather than PRO is to further reduce the amount of reorthogonalizations and recalls needed. As explained in Section 4.4.4 this comes at the price of an increased complexity of the algorithm and a heavy dependence on a free parameter η which determines the number of blocks selected for orthogonalization. Thus, MPRO was run for different values of η in the hope of establishing an optimal value (possibly dependent on the given problem). The results were however inconclusive: while the number of orthogonalizations performed was definitely less than the corresponding number for PRO, there was no dramatic decrease, and the number of recall operations was actually found to increase using MPRO for all values of η in the range considered. A recall made by MPRO is not equivalent to recalls made by PRO, however, since in many cases fewer blocks have to be scanned with MRPO, depending on η. Such scans are memory-bandwith limited which makes it hard to predict which method will come out on top.

To summarize, partial reorthogonalization is recommended for the purpose of maintaining orthogonality (or technically, semi-orthogonality) between Lanczos blocks. Preliminary results (see Section 5.3.2) indicate that there are no significant benefits from using MPRO, its performance being roughly equal to that of PRO.

8.1.3 Accelerating convergence

In Section 5.4.1 it was shown that a set of approximate eigenvectors can be used to build the Block Lanczos starting block, and accelerate the convergence of the approximated vectors. Estimates were provided for how the rate of convergence was related to the accuracy of the eigenpairs used to start the algorithm. The accuracy of an approximation were determined in terms of its tolerance as an eigenvector, defined in Eq. (5.10). The findings indicate

that in order to reduce the amount of matrix multiplications required for convergence by any appreciable amount, all of the approximate eigenvectors have to be fairly accurate. This puts restrictions on what other methods may be used to calculate these vectors. In generally it seems as if, independently of the dimensions of the matrices studied, a mean approximation tolerance in the order of $\geq 10^1$ is not better than random at all. Below that point, the benefit seems almost linear, seen with a logaritmic tolerance scale. For example, assume that it takes m multiplications to reach a precision of 10^{-9} from a random starting point, which is equal to a starting tolerance of 10^1. Then the precision has been improved by 10^{10}. If the starting point instead is 10^{-4} then the precision only needs to be improved by half as much, which means that the number of needed matrix multiplications to reach 10^{-9} is now only $\frac{m}{2}$.

Furthermore, given a particular number of available eigenvector approximations it seems as if it does not matter if each column in the starting block only contains one approximate eigenvector in each column, or if each column is a linear combination of these vectors. A reasonable guideline is choose an optimal block size for the desired number of eigenpairs in agreement with Section 8.1.1, with one exception: if the precision of the approximations only needs to be improved by a little. That is, if a number of approximations have a precision of ϵ, and a precision of $\epsilon \cdot 10^{-1}$ is needed, then a block size equal to the number of approximations seems to accelerate the convergence enough to outweigh the increased cost of the matrix-block multiplication.

8.1.4 Model spaces and computations with reduced precision

Since it has been demonstrated that the convergence can be accelerated by fitting approximate eigenvectors in starting blocks a natural follow-up question is if these approximations can somehow be obtained at a reduced cost. This was the topic of Sections 5.4.3 and 5.4.4.

In Sections 5.4.3 a matrix with single precision (sp) elements as opposed to double precision (dp), was used to calculate approximate eigenpairs. The results indicate that if the computed tolerance only needs to be in the area of 10^{-4} then one Lanczos run (with an appropriate choice of block size etc.) with the sp matrix is enough. Since each use of that matrix approximately costs half as much as a use of the dp matrix, a precision of 10^{-4} could thus be reached at half the price. Thereafter, the dp matrix needs to be used in order to increase the tolerance further, where the additional number of multiplications are proportional to the desired increase of tolerance. It seems as a tipping point is when the tolerance needs to be increased to around 10^{-10}, when the combined cost of the initial run with the sp matrix and the subsequent run with the dp matrix, ended up costing roughly the

same a single run with only the dp matrix. It thus seems advantageous to use the sp matrix as long a final tolerance $\geq 10^{-10}$ is needed.

If only the eigenvalues were of interest it was also demonstrated that the relative error could be lowered considerably by introducing just one extra multiplication with the corresponding dp matrix.

In Section 5.4.4, approximations of eigenvectors were obtained from smaller model spaces. These spaces describe the same physical system as the larger space, but with a reduced energy cut-off $N_{\max}$ resulting in a smaller matrix representing the Hamiltonian. From a physical viewpoint it seems reasonable that eigenvectors of these model spaces might serve as approximations to the desired eigenvectors if they are projected onto the larger space. However, it was found that the correlation between the eigenpairs of the model spaces was fairly low in the range of model spaces that was examined[1], in particular the residual norm of a projected eigenpair almost never dropped below 10^0. Furthermore, using the projected eigenvectors to start a new Lanczos run in the larger space reduced the number of matrix multiplications only slightly, depending on the desired tolerance. Still, since the cost of using the smaller model space matrix is just a fraction of that of the larger one, this is almost always favorable.

However, in these cases it does not seem to be much point in computing the eigenpairs of the larger spaces to any higher tolerance, as a low precision of the projected vector of the smaller matrices indicates that the eigenpairs are still far from converging to their true physical values. According to Fig. 5.17 the eigenvector of the smaller model space will not get a better residual precision in the higher model space, independent of its precision in the smaller space. From our current data it is hard to tell when this is going to happen as the eigenpairs do not show any obvious signs of convergence, considering the rate of which the eigenvalues change between the first six model spaces, as seen in Fig. 5.16. It seems as if the initial trend holds; at least 5–6 additional spaces are needed for the eigenvalues to stabilize, which would mean matrices with dimension in the order of 10^9. However this is probably to early to predict. At the point where it starts to converge the precision of the smaller space approximations is hopefully much better, which also would increase the usefulness of this method considerably.

8.1.5 Choosing an existing eigensolver

Now that the advantages and disadvantages of the various features of Lanczos-based methods have been established and compared through the lens of the

[1]The biggest model spaces had a dimension $n \approx 300\,000$.

underlying quantum mechanical many-body problem, there is still the matter of the implementation left. Since there are currently a number of libraries and routines available for the express purpose of solving LSSEs the question is to what degree this software matches our specific list of requirements. A survey of the available software can be found in Section 5.5 with a table detailing the different aspects of the existing implementations.

Based on the conclusions drawn thus far, an eigensolver based on the Block Lanczos algorithm seems to be the best option which immediately excludes ARPACK and LANCZOS. Furthermore a prospective eigensolver has to be able to run in parallel in order to efficiently handle large-scale computations thus eliminating LASO from the three remaining eigensolvers. This leaves IRBLEIGS and BLZPACK, but IRBLEIGS, while an intriguing option, is currently only available in MATLAB which comes with a lot of restrictions and an expensive license.

Hence, by principle of elimination BLZPACK is the only viable option, but there are stronger arguments in favor of it as well. BLZPACK uses modified partial reorthogonalization which significantly reduces the amount orthogonalizations required according to the investigation in Section 5.3.2. Other investigations [23], [24] give positive accounts of this approach to restoring orthogonality as well. The restarting scheme used by BLZPACK is fairly simple: after the maximum number of iterations is reached, the algorithm restarts using the unconverged Ritz vector to build a new starting block. The block size remains unaltered between different runs and Ritz vectors are passed as linear combinations to a new starting block if their numbers exceed the block size, which is consistent with the discussion in Section 8.1.3. If a restart is required, converged Ritz vectors are stored and banished during subsequent runs using external selective reorthgonalization, which is a scheme for monitoring the return of components along the converged Ritz vector, see [16].

8.1.6 Suggestions on areas for further study

Although a lot of ground has been covered concerning Lanczos methods the study has been far from comprehensive. The purpose of this section is to emphasize a few areas of interest that for various reasons were not included in this study.

One interesting possibility is a Block Lanczos solver using selective reorthogonalization (SO) which was briefly discussed in Section 4.4.5. The main reason for not including this option in the numerical tests was simply a lack of time. The vector version of this scheme is discussed at length in [17] but no papers detailing how to properly extend SO to the block case were found. Instead of attempting to work out the details, other areas of investigation

were prioritized. SO is an attractive option for restoring orthogonality since it involves orthogonalization against only a few selected Ritz vectors. This of course necessitates computing Ritz vectors at intervals but in practice these are made to coincide with check for converge which reduces the cost somewhat [17]. Since eigenstates are required for many quantum mechanical applications it would not be an actual waste to periodically compute the Ritz vectors. There is however one caveat to this approach: as demonstrated in Section 5.2, operating with blocks instead of a single vector accelerates convergence in many situations, especially in combined with other techniques such as an implicit restart [13]. This causes Ritz vectors to converge faster and since losses of orthogonality are especially pronounced in these directions it is possible that reorthogonalizations become more frequent [25]. In the end, numerical results will have to decide whether SO is worth considering.

Another area which was somewhat neglected in this project is implicit restarting schemes. These have been thoroughly investigated and are known to accelerate convergence in the case of the vector Lanczos algorithm[2]. In more recent years however, this approach has been extended to the block case [13] resulting in the MATLAB code IRBLEIGS. Since an accelerated convergence would be a desirable feature for any eigensolver this approach is certainly worth looking in to.

There are also a number of technical issues left to solve, e.g. writing a parallelized Block Lanczos code that is able to take full advantage of modern computers. According to the survey in Section 5.5 existing implementations generally do not allow for the user to control the computation of the matrix products (the exception being the sparse multiplication with the large matrix A) which is needed for an efficient parallelization.

[2]Notable implementations include ARPACK

8.2 Hamiltonian matrix properties

In Chapter 7 several investigations made on the properties of the matrices generated using the code NCSMb were presented. The investigations are important in order to understand the properties of the matrices and providing guidance to design an optimal eigensolver based on the Lanczos algorithm. Since the eigensolver will be implemented specifically to diagonalize these matrices, the code can be adapted accordingly, if this will increase efficiency further. Below follows a summary and discussion of the results obtained.

8.2.1 Dimension for different model spaces

In Section 7.1.1 investigations were made concerning how the dimension of a matrix generated by NCSMb grows. It was found that the dimension is mainly influenced by the value of the cut-off energy $N_{\max}$, and only up until a certain value does the dimension grow with N_{boson}. This is due to that the bosons of the system will have a certain number of available many-particle states for a fixed value of $N_{\max}$, and when these have been constructed, the additional bosons added will not influence the states, but merely take place in the ground state.

8.2.2 Number of nonzero elements and storage size

The number of nonzero elements N_{nz} of the matrix was investigated in Section 7.1.2 and found to increase as $N_{\text{nz},2} = d^{3/2}$ and $N_{\text{nz},3} = d^{7/4}$, in the case of two- and three-body forces, respectively, where d is the dimension of the matrix. The larger exponent for $N_{\text{nz},3}$ was expected as three-body forces allows for more connections between many-particle states, thus resulting in more nonzero elements. Using this information the memory size needed to store the three arrays defining the matrix: `values`, `columns` and `rowIndex`, could be found. From the example in Section 7.1.3 it was seen that with a storage space of 1 TB a larger matrix could be stored in the case of two-body forces than with three-body forces. The values of storage space were calculated with the assumption that the values of the indices in the arrays `column` and `rowIndex` each take up four bytes. With very large dimensions the values might instead take up eight bytes, but this can be neglected since the number of nonzeros will then be several orders of magnitude larger than the dimension, $N_{\text{nz}} \gg d$.

8.2.3 Number of updates per element

Section 7.2.1 investigated the number of updates made to each element during computation of the matrix. A figure presented the average number of updates, and it can be seen to decrease rapidly with the dimension d. The vast majority of the matrix elements were updated only once or twice, with few elements having up to ten or more updates. At a dimension of $d = 10^4$ the average number of updates was below 0.1 for both two- and three-body forces, considering values of $N_{\text{boson}} > 2$ and $N_{\text{boson}} > 4$ respectively. This motivated the implementation of a hash table for storage and lookup during calculation of the matrix, which made computations more efficient.

8.2.4 Values of matrix elements

Given all the nonzero elements of a matrix generated using NCSMb, Section 7.2.2 examined their absolute values and arranged them according to order of magnitude. It was found that the values of the elements basically are split into three groups: there is one concentration of values around 10^{-15}, and two other ones at approximately 10^{-2} and 10^{3}, respectively. The values at 10^{-15} most probably exist due to round-off errors, meaning that two terms which should have cancelled instead gave a small contribution.

8.2.5 Distribution of matrix elements

In terms of the distribution of the nonzero elements over the matrix, in Section 7.2.3 it could be seen that in some parts of the matrix the nonzeros dominated, while other parts mainly contained zeros. How the elements are distributed is important when it comes to the many matrix-vector multiplications that will be performed in the implementation of an eigensolver. Using parallel kernels, the matrix can be split into several pieces to speed up calculations, and thus the matrix-vector multiplications will also be split. The question then arises how the split should be made.

If the matrix is to be split in equally large parts, the nonzero and zero elements should be approximately evenly distributed over the matrix, in order for calculations to be load-balanced and efficient. This was found through the work of Sternberg et al. [26], i.e. computations made on large, sparse matrices were found to be much more efficient with an even distribution of elements over the matrix. The even distribution was achieved by rearranging the order of the many-particle-basis along the rows of the matrix, where they initially were ordered lexicographically.

In terms of cache, it will be advantageous with large areas of the matrix containing zeros or nonzeros, as this will allow calculations to be performed

more effectively. Memory caches thrive on microclusterisation. On the other hand, this assumes that the matrix can be split into blocks of unequal sizes and distributed among the kernels. Otherwise, if some blocks contain many more nonzeros than others, some CPUs will end up waiting for others to complete their computations, which is far from ideal.

8.2.6 Improvements of the code

The code NCSMb is not a finished product and still contains areas of possible improvements. One problem that has now been identified and solved was that the program allocated memory for many more single-particle states than was actually needed, which was illustrated in Fig. 7.8. This greatly restricted the size of the model space that could be used. The implementation of an additional hash table improved the execution time. Furthermore, the sparse matrix storing format CSR reduced the storage size significantly for the matrices. Another area of improvement is of course the implementation of an eigensolver, and suggestions on how this should be done are summarized in the next chapter.

8.2.7 Limitations

The matrices generated using the code NCSMb were limited in dimension due to the time taken for computations and the storage size. However, the matrix properties that were analyzed are representative for larger dimensions as well, and therefore our results will generally hold. When it comes to the interaction parameters E_2 and E_3 no realistic values were known, but the results obtained and e.g. the general distribution of the matrix are expected to follow the same pattern for other values of these parameters.

9 Recommendations for a future implementation

The aim of this final chapter is to summarize the discussion from the previous chapter and provide a specific list of requirements the future implementation of an eigensolver. Based on these criteria, an available eigensolver that can be used immediately will also be recommended.

- The eigensolver should be based on the Block Lanczos algorithm. When the sparse matrix multiplication is expensive the block size should be set to $p = 1$ or $p = 2$ for one respectively two eigenpairs, otherwise $p = 4$. For more than ten eigenpairs $p = 8$ seems to work equally well. [1]

- A reorthogonalization technique that attempts to minimize the number of orthogonalizations and recalls should be used. For these purposes, partial reorthogonalization (or the modified version of this scheme) is recommended. It is also possible to use selective reorthogonalization which was not investigated but is believed to be a viable option.

- When out of memory, the algorithm should perform a restart using unconverged Ritz vectors to form a new starting block. In general, the block size p should always equal one of the recommended sizes and if the number of available Ritz vectors exceeds p some of them may be passed as a linear combination to the new starting block. Converged Ritz vectors may be stored away after the completion of a Lanczos run and banished during subsequent runs using a suitable technique, e.g. external selective reorthgonalization.

- Unless extremely accurate eigenpairs need to be computed, it is possible to use a single-precision matrix to either directly calculate the eigenpairs or to calculate approximate eigenvectors which can then be used to start a new Lanczos run on the double-precision matrix.

- Approximate eigenstates may be calculated from a smaller model space and projected onto a larger space and used to start the Lanczos run resulting in a somewhat faster convergence. Although no significant

[1] Based on the matrices available for investigation, were the biggest matrix were of dimension 300 000.

decrease in the required number of matrix multiplications was observed it is believed that the benefits of this approach will increase when larger model spaces are considered.

- Of the software currently available, BLZPACK is the best match to this list of requirements. It is based on a Block Lanczos recursion, uses modified partial reorthgonalization and is able to run in parallel using MPI.

- During the parallel implementation the demands on the distribution of the matrix elements will differ depending on if the matrix is to be split into part of uneven sizes, or equally large parts. In order to as effectively as possible use the cache, the columns and rows of the matrix can be rearranged appropriately.

References

[1] Sherrill, C. D., Schaefer III, H. F. (1999) *The Configuration Interaction Method: Advances in Highly Correlated Approaches*. San Diego: Academic Press. pp. 143-269. (Advances in Quantum Chemistry, vol. 34, pp. 143-269)

[2] Tölle, S., *Private communication*.

[3] Dickhoff, W. H., Van Neck, D., (2005) *Many-Body Theory Exposed!*, Singapore: World Scientific Publishing Co. Pte. Ltd.

[4] Negele, J. W., Orland, H, (1998) *Quantum Many-particle Systems*, Westview Press.

[5] Bruus, H., Flensberg, K., (2004) *Many-body quantum theory in condensed matter physics: an introduction*, Oxford University Press.

[6] Caurier, E., Martinez-Pinedo, G., Nowacki, F., Poves, A., Zuker, A. P. (2005) The shell model as a unified view of nuclear structure, *Reviews of Modern Physics*, vol. 77, p.427.

[7] Navrátil, P., Quaglioni, S., Stetcu, I., Barrett, B. R. (2009) Recent developments in no-core shell-model calculations. *J. Phys. G: Nucl. Part. Phys.*, vol. 36, no. 8

[8] Barrett, B. R., Navrátil, P., Nogga, A., Ormand, W. E., Stetcu, I., Vary, J. P., Zhan, H. (2005) *Ab Initio* Large-Basis No-Core Shell Model. *AIP Conf. Proc.*, vol. 769, pp. 1257-1262

[9] Heath, M. T. (2002) *Scientific Computing : An Introductory Survey*, Second Edition. New York, McGraw-Hill.

[10] Underwood, R. (1975) *An interativ Block Lanczos method for the solutions of large sparse symmetric eigenproblems*. Stanford: Standford University (Ph.D. thesis, Department of computer science).

[11] (1978) *Error analysis of the Lanczos algorithm for tridiagonalizing a symmetric matrix*. J. Inst. Math. Appl., 18 , pp. 341 - 349.

[12] Stewart, G. (2001) *Matrix Algorithms, Volume II: Eigensystems*. Philadelphia: Society for Industrial and Applied Mathematics.

[13] Baglama, J., Calvetti, D., and Reichel, L. *IRBL: An implicitly restarted Block-Lanczos method for large-scale hermitian eigenproblems.* SIAM J. Sci. Comput., vol 24, no. 5, pp. 1650-1677.

[14] Simon, H. (1984) *Analysis of the Symmetric Lanczos Algorithm with Reorthogonaliration Methods.* Linear Algebra and its applications, vol. 61, pp. 101-131.

[15] Cullum, J. and Willoughby, R. (1985) *Lanczos algorithms for large symmetric eigenvalue computations, Vol I: Theory.* Boston: Birkhäuser Boston.

[16] Grimes, R., Lewis, J. and Simon, H. (1988) *The Implementation of a Block Lanczos algorithm with reothogonalization methods.* Moffet Field, CA: NASA Amnes Research Center.

[17] Parlett B. and Scott D. (1979). *The Lanczos algorithm with selective orthogonalization.* Mathematics of Computation, vol. 33, no. 145, pp. 217-238.

[18] Golub, H. and Van Loan, C. (1996) *Matrix computations.* Third edition. Baltimore, Maryland: The Johns Hopkins University Press.

[19] Saad, Y. (1980) *On the Rates of Convergence of the Lanczos and the Block-Lanczos Methods* SIAM Journal on Numerical Analysis, vol. 17, no. 5, pp. 687-706.

[20] Johansson, H., *Private communication.*

[21] Intel Corporation, (2011) *Sparse Matrix Storage Formats* , `http://software.intel.com/sites/products/documentation/hpc/compilerpro/en-us/cpp/win/mkl/refman/appendices/mkl_appA_SMSF.html#mkl_appA_SMSF_2`, (2012-03-09)

[22] Loudon, K. (1999) *Mastering Algorithms with C*, Sebastopol: O'Reilly Media.

[23] Simon, H. (1984) *The Lanczos Algorithm With Partial Reorthogonalization.* Mathematics of Computation, vol. 42, no. 165, pp. 115-142.

[24] Sanzheng, Q., Liu, G., Xu, W. (2005), *Block Lanczos tridiagonalization of complex symmetric matrices* Advanced Signal Processing Algorithms, Architectures, and Implementations XV Proceedings of the SPIE, vol. 5910, pp. 285-295.

[25] Grimes, R., Lewis, J. and Simon, H. (1991) *A Shifted Block Lanczos Algorithm for Solving Sparse Symmetric Generalized Eigenproblems.* Moffet Field, CA: NASA Amnes Research Center.

[26] Sternberg, P. et al., (2008), Accelerating Configuration Interaction Calculations for Nuclear Structure *SC '08 Proceedings of the 2008 ACM/IEEE conference on Supercomputing*; 2008, Austin, Texas, USA.

[27] Parlett, B.N. (1980) *The Symmetric Eigenvalue Problem*. Englewood Cliffs, N.J: Prentice-Hall. (Prentice-Hall Series in Computational Mathematics), secs. 11-13.

Appendices

A Jacobi Coordinates

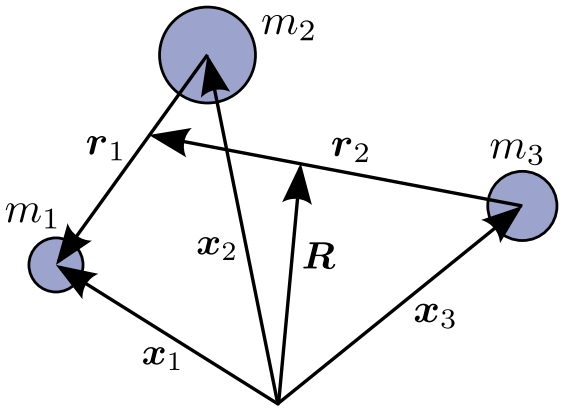


Figure A.1. *The Jacobi coordinates for a three-body system.*

Jacobi, or *relative*, *coordinates* are often used for many-particle systems to simplify calculations. Consider a many-particle system with coordinates $\boldsymbol{x}_i$ relative to origin. Their positions can also be represented by the Jacobi coordinates $\boldsymbol{r}_i$, and the center of mass coordinate $\boldsymbol{R}$. Starting with a system of two particles with masses m_1 and m_2, their positions can be given by the Jacobi coordinates

$$\boldsymbol{r}_1 = \boldsymbol{x}_1 - \boldsymbol{x}_2, \qquad \boldsymbol{R} = \frac{1}{m_1 + m_2}(m_1\boldsymbol{x}_1 + m_2\boldsymbol{x}_2). \tag{A.1}$$

The Jacobi coordinates for the third particle in a three-body system is obtained by using the relative coordinates between the third particle and the center of mass of the first two particles, giving

$$\begin{aligned} \boldsymbol{r}_1 &= \boldsymbol{x}_1 - \boldsymbol{x}_2\,, \\ \boldsymbol{r}_2 &= \frac{1}{m_1 + m_2}(m_1\boldsymbol{x}_1 + m_2\boldsymbol{x}_2) - \boldsymbol{x}_3\,, \\ \boldsymbol{R} &= \frac{1}{m_1 + m_2 + m_3}(m_1\boldsymbol{x}_1 + m_2\boldsymbol{x}_2 + m_3\boldsymbol{x}_3). \end{aligned} \tag{A.2}$$

A three-body system with its Jacobi coordinates can be seen in Fig. A.1.

The Jacobi coordinates for an N-body system are

$$\begin{aligned} \boldsymbol{r}_1 &= \boldsymbol{x}_1 - \boldsymbol{x}_2 \,, \\ \boldsymbol{r}_i &= \frac{1}{m_1 + m_2 + ... + m_i} \sum_{k=1}^{i} m_k \boldsymbol{x}_k - \boldsymbol{x}_{i+1} \,, \\ \boldsymbol{R} &= \frac{1}{M} \sum_{k=1}^{N} m_k \boldsymbol{x}_k \,, \end{aligned} \tag{A.3}$$

where $2 \leq i \leq N-1$ and where M is the total mass of all particles. For N identical particles inside a HO potential it is convenient to use normalized Jacobi coordinates, given by

$$\boldsymbol{r}_i = \frac{1}{\sqrt{(n+1)n}} (\boldsymbol{x}_1 + \boldsymbol{x}_2 + \ldots + \boldsymbol{x}_i - i\boldsymbol{x}_{i+1}), \tag{A.4a}$$

$$\boldsymbol{R} = \frac{1}{\sqrt{N}} (\boldsymbol{x}_1 + \boldsymbol{x}_2 + \ldots + \boldsymbol{x}_N). \tag{A.4b}$$

B The General Behavior of the Lanczos Algorithm

This is a discussion of the general behaviour of the Lanczos Algorithm, and the mechanisms of which it identifies and and singles out the most extreme eigenpairs. For a more complete proof of the convergence of Lanczos, see [27].

The Lanczos algorithm converges first towards the eigenpairs of endpoint eigenvalues, unrelated of their magnitude. For example, imagine a symmetric matrix A with eigenvalues evenly spread out from -10 to 100. Lanczos will then find -10 and 100 first, instead of the eigenvalues with the greatest or smallest magnitudes. One way to better understand this is to carefully study how the succeeding Lanczos vectors q_i, of Section 4.1, are formed. The critical question to keep in mind is how the Krylov subspace $\mathcal{K}_k$ with matrices Q_k of $Q = [Q_k \; U_{n-k}]$ and T_k can carry all the important information about the extreme eigenpairs even for $k \ll n$.

In the basic Lanczos algorithm new Lanczos vectors are computed from the relation in Eq. (4.14). What this does is basically to generate a new vector Aq_i and then subtract the components of the two preceding vectors from it, to finally normalize it. Another way to put this is by the relation

$$q_{i+1} = \frac{1}{\beta_i}\left[Aq_i - (q_i^T A q_i)q_i - (q_{i-1}^T A q_i)q_{i-1}\right], \tag{B.1}$$

with q_0 defined to be the zero vector.

If $(v_j)_{j=1}^n$ is the set of eigenvectors of the symmetric matrix A, and $(\lambda_j)_{j=1}^n$ the corresponding eigenvalues, any vector q_i, $i \geq 1$, can be expressed as

$$q_i = \sum_{j=1}^{n} c_{ij} v_j \; , \tag{B.2}$$

where the eigenvectors satisfy

$$v_i^T v_j = \delta_{ij} \; , \tag{B.3}$$

and the real coefficients c_{ij} fulfill

$$\sum_{j=1}^{n} c_{ij}^2 = 1, \tag{B.4}$$

as $q_i^T q_j = \delta_{ij}$. Expressed in this way, as a sum of eigenvectors, a multiplication by A is easily expressed as

$$Aq_i = \sum_{j=1}^{n} c_{ij} A v_j = \sum_{j=1}^{n} c_{ij} \lambda_j v_j \ . \tag{B.5}$$

Since $q_0 = 0$, q_1 can be arbitrarily chosen, as long as it is normalized. The subsequent vector q_2 can then according to (B.1) be formed, giving

$$\begin{aligned} q_2 &= \frac{1}{\beta_1} \left[Aq_1 - \alpha_1 q_1 \right] = \frac{1}{\beta_1} \sum_{j=1}^{n} \left(c_{1j} \lambda_j v_j - c_{1j} \alpha_1 v_j \right) \\ &= \frac{1}{\beta_1} \sum_{j=1}^{n} c_{1j} \left(\lambda_j - \alpha_1 \right) v_j = \sum_{j=1}^{n} c_{2j} v_j \ , \end{aligned} \tag{B.6}$$

where c_{2j} is given by

$$c_{2j} = \frac{\lambda_j - \alpha_1}{\beta_1} c_{1j} \ , \tag{B.7}$$

and $\alpha_1 = q_1^T A q_1$. Expressed in the sum notation, using $v_j^T v_{j'} = \delta_{jj'}$, this becomes

$$\alpha_1 = \left(\sum_{j=1}^{n} c_{1j} v_j \right)^T \left(\sum_{j'=1}^{n'} c_{1j'} \lambda_{j'} v_{j'} \right) = \sum_{jj'}^{nn'} \lambda_{j'} c_{1j} c_{1j'} v_j^T v_{j'} = \sum_{j=1}^{n} \lambda_j c_{1j}^2 \, , \tag{B.8}$$

because $v_j^T v_{j'} = \delta_{jj'}$.

Furthermore, using that $\lambda_{\min} \leq \lambda_j \leq \lambda_{\max}$ together with (B.4) will give

$$\alpha_1 \geq \lambda_{\min} \sum_{j=1}^{n} c_{1j}^2 = \lambda_{\min}, \tag{B.9}$$

and

$$\alpha_1 \leq \lambda_{\max} \sum_{j=1}^{n} c_{1j}^2 = \lambda_{\max}. \tag{B.10}$$

α_1 is apparently a number between $\lambda_{\min}$ and $\lambda_{\max}$[1], which also holds for any α_i. One way to look at α_i is as a weighted mean value of all the eigenvalues. For example, if every term in the sum has a position λ_j, and a mass c_{ij}^2, then α_i simply denotes the center of mass. According to Eq. (B.7) it can then be concluded that coefficients with eigenvalues close to this mean value α_i will

[1] $\lambda_{\max/\min}$ denotes the largest or least eigenvalue of an eigenvector with a nonzero coefficient. If there are more extreme eigenvalues, but with a zero coefficient, this eigenvalue is ignored as it is invincible to the algorithm at all times.

tend to shrink in magnitude, as opposed to those far away which will grow. To say a little bit more β_1 must be found.

Due to that β_1 is the normalizing factor of q_2 it is known from Eq. (B.6) that

$$\beta_1^2 = \left(\sum_{j=1}^{n} c_{1j}(\lambda_j - \alpha_1)v_j \right)^2 = \sum_{j=1}^{n} c_{1j}^2.(\lambda_j - \alpha_1)^2 \tag{B.11}$$

Thus, β_1 is the 2-norm of a weighted mean distance from α_1

$$\beta_1 = ||(\lambda - I_n\alpha_1)c_1||_2 \,, \tag{B.12}$$

where c_1 denotes a vector with the coefficients $(c_{1j})_{j=1}^n$, λ denotes the diagonal eigenvalue matrix and I_n is the $n \times n$ identity matrix. If α_1 is thought of as a center of mass, β_1 can be thought of as the mean distance of the mass from the mass center. That means according to Eq. (B.7) that if an eigenvalue is further from the mean value than the average distance, then that coefficient will grow, and to the contrary, coefficients that are closer will shrink. This first step thus enriched q_2 in components of the extreme eigenpairs.

Taking another step forward, from q_2 to q_3, things will however start to look a little different, since more terms are introduced:

$$\begin{aligned} q_3 &= \frac{1}{\beta_2}\left(Aq_2 - \alpha_2 q_2 - \beta_1 q_1\right) \\ &= \frac{1}{\beta_2}\sum_{j=1}^{n}\left(c_{2j}\lambda_j - c_{2j}\alpha_2 - c_{1j}\beta_1\right)v_j \,. \end{aligned} \tag{B.13}$$

With the use of Eq. (B.7) this becomes

$$q_3 = \frac{1}{\beta_2}\sum_{j=1}^{n}\left(\frac{(\lambda_j - \alpha_1)(\lambda_j - \alpha_2)}{\beta_1} - \beta_1\right)c_{1j}v_j = \sum_{j=1}^{n} c_{3j}v_j \,, \tag{B.14}$$

where the coefficient relation for c_{3j} is

$$c_{3j} = \left(\frac{(\lambda_j - \alpha_1)(\lambda_j - \alpha_2)}{\beta_1\beta_2} - \frac{\beta_1}{\beta_2}\right)c_{1j}. \tag{B.15}$$

α_2 was formed in the same way as α_1 and will – if we return to the analogy with a center of mass – be the center of mass of the new distribution. Remember that the first step favored the elements that were far from α_1. That means that if for example α_1 is much closer to $\lambda_{\min}$ than $\lambda_{\max}$, then the coefficients of the eigenvalues closest to $\lambda_{\max}$ would have grown the most.

This tilts the new center of mass towards $\lambda_{\max}$, and α_2 would thus take a jump closer to $\lambda_{\max}$, of a size proportional to the earlier deviation of ${\alpha_1}^2$.

To see what will happen to β_2 we use the fact that q_3 is normalized

$$\begin{aligned}\beta_2^2 &= \sum_{j=1}^{n}\left(\frac{(\lambda_j-\alpha_1)(\lambda_j-\alpha_2)}{\beta_1}-\beta_1\right)^2 c_{1j}^2,\\ &= \beta_1^2\sum_{j=1}^{n}\left(\frac{(\lambda_j-\alpha_1)(\lambda_j-\alpha_2)}{\beta_1^2}-1\right)^2 c_{1j}^2 = \beta_1^2\sum_{j=1}^{n}X_{2j}^2 c_{1j}^2. \qquad \text{(B.16)}\end{aligned}$$

β_2 cannot be a measure of the mean deviation away from α_2 anymore. To analyse this the sum must be examined more carefully. To begin with, if every term X_{2j}^2 would be equal to some positive constant σ^2, then $\beta_2^2 = \sigma^2\beta_1^2$, using Eq. (B.4). X_{2j} is not a constant, but it does tell us that if many of the terms $(X_{1j})_{j=1}^n$ are greater than one, it is more likely that β_2 will be greater than β_1. There are two cases when $X_{2j}^2 > 1$: $X_{2j} < -1$ and $X_{2j} > 1$. In the first case,

$$X_{2j} < -1 \;\Rightarrow\; (\lambda_j-\alpha_1)(\lambda_j-\alpha_2) < 0. \qquad \text{(B.17)}$$

This means that if j fullfills

$$\min(\alpha_1,\alpha_2) < \lambda_j < \max(\alpha_1,\alpha_2) \qquad \text{(B.18)}$$

then $X_{2j}^2 > 1$. In the second case,

$$X_{2j} > 1 \;\Rightarrow\; (\lambda_j-\alpha_1)(\lambda_j-\alpha_2) > 2\beta_1^2. \qquad \text{(B.19)}$$

This is somewhat more difficult to evaluate. However, by using the approximation $\alpha_1 \approx \alpha_2$ and setting $\widetilde{\alpha}_{12} > \frac{\alpha_1+\alpha_2}{2}$, the following is found

$$(\lambda_j-\widetilde{\alpha}_{12})^2 > 2\beta_1^2. \qquad \text{(B.20)}$$

Thus, $X_{2j}^2 > 1$ if

$$\lambda_j < \widetilde{\alpha}_{12}-\sqrt{2}\beta_1 \quad \text{or} \quad \lambda_j > \widetilde{\alpha}_{12}+\sqrt{2}\beta_1. \qquad \text{(B.21)}$$

To sum this up; β_2 will tend to increase if the α_2-value changes a lot from α_1, or if a lot of the eigenvalues are further from $\widetilde{\alpha}_{12}$ than $\sqrt{2}\beta_1$.[3] The first will happen if the initial weighted eigenvalues strongly favored one end of the eigenvalue spectrum. The second will happen if many of the eigenvalues are further from $\widetilde{\alpha}_{12}$ than $\sqrt{2}$ times the initial mean deviation away from α_1.

[2]In this context a deviation just means that the effect of a mean value placement will force the next center of mass to shift.

[3]Remember that β_1 still is a measure of the initial deviation of the weighted eigenvalues, away from its "center of mass".

It seems as though if the initial distribution $(c_{1j})_{j=1}^{n}$ is asymmetrical, or have most of the "weights" on eigenvectors with eigenvalues somewhere in the middle of the spectrum, β_2 will tend to increase, while on the other hand, if the initial distribution is symmetrical or have most of its "weights" at the endpoint eigenvalues, this will tend to decrease β_2. It should be pointed out that this only applies to eigenvalues that do have a weight. If a coefficient of an eigenvector at any point becomes zero it will not contribute anymore to any α or β value.

Finally, considering q_3, how will the coefficients change according to Eq. (B.15)? Let us set $\beta_2 = \sigma_1\beta_1$, $\sigma_i > 0$, and examine the case when $c_{3j}^2 > 1$. We have,

$$c_{3j}^2 = \frac{1}{\sigma_1^2}\left(\frac{(\lambda_j - \alpha_1)(\lambda_j - \alpha_2)}{\beta_1^2} - 1\right)^2 c_{1j}^2 = \left(\frac{X_{2j}}{\sigma_1}\right)^2 c_{1j}^2\,. \tag{B.22}$$

This is advantageous, since it is already known from the studies of β_2 how X_{2j} behaves, only σ_1 must be taken into account. In the first case,

$$\frac{(\lambda_j - \alpha_1)(\lambda_j - \alpha_2)}{\beta_1^2} - 1 < -\sigma_1, \tag{B.23}$$

and with some calculation

$$\lambda_j > \frac{\alpha_1 + \alpha_2}{2} - \sqrt{\left(\frac{\alpha_1 - \alpha_2}{2}\right)^2 + (1 - \sigma_1)\beta_1^2}, \quad \text{or} \tag{B.24}$$

$$\lambda_j < \frac{\alpha_1 + \alpha_2}{2} + \sqrt{\left(\frac{\alpha_1 - \alpha_2}{2}\right)^2 + (1 - \sigma_1)\beta_1^2}. \tag{B.25}$$

From these conditions it can be seen that the smaller σ_1 is, the wider the span gets. On the contrary, when σ_1 grows larger the span lessens, until the point where it disappears. If $\alpha_1 = \alpha_2$ for instance, this happens when $\sigma_1 = 1$.

For the second case, we have

$$\frac{(\lambda_j - \alpha_1)(\lambda_j - \alpha_2)}{\beta_1^2} - 1 > \sigma_1\,. \tag{B.26}$$

As before, it can be approximated that α_1 and α_2 may be exchanged for $\frac{\alpha_1+\alpha_2}{2}$ [4] and this yields

$$\lambda_j > \frac{\alpha_1 + \alpha_2}{2} + \sqrt{\sigma_1 + 1}\,\beta_1\,, \quad \text{or} \quad \lambda_j < \frac{\alpha_1 + \alpha_2}{2} - \sqrt{\sigma_1 + 1}\,\beta_1\,. \tag{B.27}$$

[4] This may of course be more or less accurate. However these results will make an impact on the outer ends of the eigenvalue spectrum, and seen from that perspective the differences between the two α_i should not matter significantly.

In this case a larger value of σ_1 means that eigenvalues must be further out on the edges to fulfill the relation. Very roughly the following can be said about c_{3j}: $c_{3j} > c_{1j}$ if λ_j are far enough on the edges, or if it is between two successive α_i. Additionally, c_{3j} will be forced towards zero if λ_j is at a distance β_1 from any α_i. If $\beta_2 > \beta_1$, then $\sigma > 1$ and this range of growth will be moved further out on the edges.

Since further steps q_{i+1} in the algorithm will show similar relations, some general behavior can now be discussed. With coefficients asymmetrically distributed, the middle point of the span will be favored, as the next center of mass will be moved closer to the middle. Therefore the α-values will soon converge approximately in the middle of the spectrum. The mean distance from the center will thus increase and coefficients need to be even further from the mass center in order for them to still grow. At some point, β will reach the most extreme values and thus reduce them to zero. This would effectively shift the load towards the center, forcing the next β to drastically become smaller, which in turn would benefit the next extreme eigenvalues as they suddenly again are far enough from α[5], thus restarting the process, increasing the next eigenvalues and taking them out, moves the load to the center again, and so forth. At the same moment a coefficient of an eigenvalue reaches zero, no more information can be extracted from it, and the Lanzos algorithm has found extracted all information of its corresponding eigenpair. Fig. B.1 shows how the magnitudes of c_{ij} changes for a matrix A with evenly distributed and weighted eigenvalues.

[5]measured with the new and shorter β.

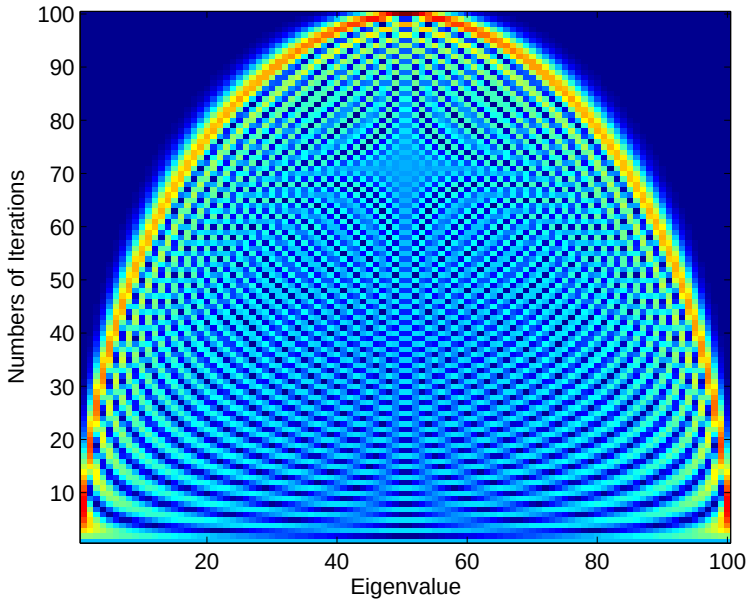


Figure B.1. *This illustrates how the coefficients to the different eigenvalues change their magnitudes depending on the iteration and the relative size of the associated eigenvalues. In this case the eigenvalues are evenly distributed and weighted.*

C Proofs of Some Basic Properties of the Lanczos Blocks

This appendix contains a proof of one of the most important properties of the Lanczos blocks, namely the fact that their columns constitute an orthonormal set of vectors. This proof can also be found in [16], but readers are referred to [10] for a more rigorous derivation of the Block Lanczos algorithm. Some of the basic relations derived in the original Block Lanczos in Chapter 4, Section 4.2.1 will be useful here, namely Eqs. (4.20) and (4.25). The orthogonality condition is expressed in matrix notation as:

$$Q_j^T Q_k = 0 \text{ for all } i \neq k. \tag{C.1}$$

The proof can now be broken down into two steps: First prove local orthogonality among neighboring blocks by induction over j, i.e. show that

$$Q_j^T Q_{j+1} = 0 \text{ and } Q_{j-1}^T Q_{j+1} = 0. \tag{C.2}$$

For $j = 1$, the orthogonality follows from equation Eq. (4.20). Next, suppose that for $j \geq 2$ Eq. (4.25) holds for all $i < j$ and premultiply the equation by Q_j^T:

$$Q_j^T Q_{j+1} B_{j+1} = Q_j^T A Q_j - Q_j^T Q_j A_j - Q_j^T Q_{j-1} B_j^T. \tag{C.3}$$

Here, the last term of the right hand side of Eq. (C.3) vanishes because of the inductive assumption. Furthermore, since $Q_j^T Q_j = I_j$ where I_j denotes the $j \times j$ unity matrix and $Q_j^T A Q_j = A_j$ by definition, the first two terms cancel thus implying that the blocks are indeed orthogonal. Similarily, premultiplying equation Eq. (4.25) by Q_{j-1}^T yields

$$Q_{j-1}^T Q_{j+1} B_{j+1} = Q_{j-1}^T A Q_j - Q_{j-1}^T Q_j A_j - Q_{j-1}^T Q_{j-1} B_j^T. \tag{C.4}$$

This time the second term vanishes on account of the inductive assumption while the first and third terms cancel since $Q_{j-1}^T Q_{j-1} = I_{j-1}$ and

$$Q_{j-1}^T A Q_j = B_j^T. \tag{C.5}$$

The bonus result Eq. (C.5), can be obtained by premultiplying Eq. (4.25) with Q_{j-1}^T:

$$Q_{j-1}^T Q_{j+1} B_{j+1} = Q_{j-1}^T A Q_j - Q_{j-1}^T Q_j A_j - Q_{j-1}^T Q_{j-1} B_j^T. \tag{C.6}$$

Here $Q_{j-1}^T Q_{j+1} = 0$ and $Q_{j-1}^T Q_j = 0$ by local orthogonality and thus Eq. (C.6) reduces to Eq. (C.5). This concludes the proof of local orthogonality among the Lanczos blocks. Finally, it remains to prove global orthogonality among the blocks, i.e.

$$Q_i^T Q_{j+1} = 0 \text{ for } i = 1, \ldots, j-1. \tag{C.7}$$

Proceeding again by induction over j, for $j = 1$ it follows that $Q_1^T Q_2 = 0$ from local orthogonality. Now, for $j \geq 2$ assume that global orthogonality holds for all $i < j$ and premultiply equation Eq. (4.25) by Q_i^T to obtain

$$Q_i^T Q_{j+1} B_{j+1} = Q_i^T A Q_j - Q_i^T Q_j A_j - Q_i^T Q_{j-1} B_j^T. \tag{C.8}$$

The last two terms on the right hand side of Eq. (C.8) vanish from the inductive assumption and the remaining term $Q_i^T A Q_j$ can be rewritten by taking the transpose of Eq. (4.25) to obtain an expression for $Q_i^T A$:

$$Q_i^T A = B_{j+1}^T Q_{j+1}^T + A_j Q_j^T + B_j Q_{j-1}^T. \tag{C.9}$$

Combining this with Eq. (C.8) finally yields

$$Q_j^T A Q_i = \left(B_{j+1}^T Q_{j+1}^T + A_j Q_j^T + B_j Q_{j-1}^T \right) Q_i = 0, \tag{C.10}$$

where the entire right hand side dropped out because of the inductive assumption. This concludes the proof of total orthogonality among the Lanczos blocks.

D Links to Source Codes

This appendix provides links to file repositories containing source code and documentation for the various eigensolvers discussed in Section 5.5.

IRBLEIGS: `http://www.math.uri.edu/~jbaglama/#Software`

ARPACK/P_ARPACK: `http://www.caam.rice.edu/software/ARPACK/`

ARPACK++: `http://www.ime.unicamp.br/~chico/arpack++/`

BLZPACK: `http://crd-legacy.lbl.gov/~osni/#Software`

LASO: `http://www.netlib.org/laso/`

LANCZOS: `http://www.netlib.org/lanczos/`