Efficient Density Matrix Methods for Large Scale Electronic Structure Calculations

ANASTASIA KRUCHININA
Abstract

Efficient and accurate methods for computing the density matrix are necessary to be able to perform large scale electronic structure calculations. For sufficiently sparse matrices, the computational cost of recursive polynomial expansions to construct the density matrix scales linearly with increasing system size. In this work, parameterless stopping criteria for recursive polynomial expansions are developed. The proposed stopping criteria automatically adapt to a change in the requested accuracy, perform at almost no additional cost and do not require any user-defined tolerances.

Compared to the traditional diagonalization approach, in linear scaling methods molecular orbitals are not readily available. In this work, the interior eigenvalue problem for the Fock/Kohn-Sham matrix is coupled to the recursive polynomial expansions. The idea is to view the polynomial, obtained in the recursive expansion, as an eigenvalue filter, giving large separation between eigenvalues of interest. An efficient method for computation of homo and lumo eigenvectors is developed. Moreover, a method for computation of multiple eigenvectors around the homo-lumo gap is implemented and evaluated.

An original method for inverse factorization of Hermitian positive definite matrices is developed in this work. Novel theoretical tools for analysis of the decay properties of matrix element magnitude in electronic structure calculations are proposed. Of particular interest is an inverse factor of the basis set overlap matrix required for the density matrix construction. It is shown that the proposed inverse factorization algorithm drastically reduces the communication cost compared to state-of-the-art methods.

To perform large scale numerical tests, most of the proposed methods are implemented in the quantum chemistry program Ergo, also presented in this thesis. The recursive polynomial expansion in Ergo is parallelized using the Chunks and Tasks matrix library. It is shown that the communication cost per process of the recursive polynomial expansion implementation tends to a constant in a weak scaling setting.

Keywords: electronic structure; linear scaling; density matrix; density functional theory; Hartree-Fock; recursive polynomial expansion; density matrix purification; eigenvectors; molecular orbitals; stopping criteria; inverse factorization; matrix sparsity; parallelization

© Anastasia Kruchinina 2019

ISSN 1651-6214
urn:nbn:se:uu:diva-393387 (http://urn.kb.se/resolve?urn=urn:nbn:se:uu:diva-393387)
List of papers

This thesis is based on the following papers, which are referred to in the text by their Roman numerals.

I  *Parameterless stopping criteria for recursive density matrix expansions.*
   Anastasia Kruchinina, Elias Rudberg, and Emanuel H. Rubensson.

**Contributions:**
I was responsible for the implementation, numerical experiments and preparation of the manuscript. The ideas were developed in a close collaboration with the co-authors.

II  *On-the-fly computation of frontal orbitals in density matrix expansions.*
    Anastasia Kruchinina, Elias Rudberg, and Emanuel H. Rubensson.

**Contributions:**
I was responsible for the implementation, numerical experiments and preparation of the manuscript. The ideas were developed in a close collaboration with the co-authors.

    Elias Rudberg, Emanuel H. Rubensson, Paweł Sałek, and
    Anastasia Kruchinina.
    SoftwareX 2018, 7, 107-111
    DOI: 10.1016/j.softx.2018.03.005

**Contributions:**
I contributed to the development of the presented software and to the manuscript.

IV  *Localized inverse factorization.*
    Emanuel H. Rubensson, Anton G. Artemov, Anastasia Kruchinina, and
    Elias Rudberg.
    arXiv:1812.04919

**Contributions:**
I performed the parallel performance evaluation and contributed to the manuscript. The ideas were developed in a close collaboration with the co-authors.
V  Efficient computation of the density matrix with error control on distributed computer systems.
Anastasia Kruchinina, Elias Rudberg and Emanuel H. Rubensson.
arXiv:1909.12533

Contributions:
I was responsible for the implementation, numerical experiments and preparation of the manuscript. The ideas were developed in a close collaboration with the co-authors.

VI  Multiple eigenvectors around the homo-lumo gap as a cheap by-product in linear scaling electronic structure calculations.
Anastasia Kruchinina.
arXiv:1909.11662

Contributions:
I am the sole contributor to this work.

Reprints were made with permission from the publishers.
This paper is not discussed in the summary, but it is related to the thesis content.

- Multidimensional performance and scalability analysis for diverse applications based on system monitoring data.
Contents

1 Introduction .................................................................................................. 9
  1.1 Electronic Schrödinger equation ................................................... 11
  1.2 Self-consistent field procedure ...................................................... 12
  1.3 Decay properties of matrix element magnitude ........................... 14

2 Density matrix construction ...................................................................... 17
  2.1 Inverse factors of the basis set overlap matrix ............................. 18
  2.2 Overview of density matrix methods ............................................ 20
  2.3 Recursive polynomial expansions ................................................. 23
  2.4 Parameterless stopping criteria ...................................................... 29
  2.5 Molecular orbitals around the homo-lumo gap ............................ 32

3 Density matrix construction on distributed-memory computer clusters 36
  3.1 Parallel programming models ....................................................... 36
    3.1.1 Chunks and Tasks programming model ................................. 38
  3.2 Chunks and Tasks matrix library ................................................... 39
  3.3 Parallel recursive polynomial expansion ...................................... 41

4 Thesis contributions ................................................................................... 42

5 Summary in Swedish ................................................................................. 44

6 Acknowledgments ..................................................................................... 46

References ........................................................................................................ 47
Quantum chemistry considers nuclei and electrons as basic building blocks of atoms and molecules, and attempts to explain the chemical structure and chemical reactions using fundamental laws of physics. With growing capabilities of computers, the computational modeling of natural processes becomes an attractive complement to the physical experiment. Moreover, computer simulations facilitate the study of unstable compounds with short lifetime. If the structure of nuclei and electrons, and all forces acting between them, can at any moment of time be represented in a computer program, the flow of chemical reactions can be simulated almost exactly. Unfortunately, such predictions are extremely complex and, in general, not feasible with current technologies. Computational quantum chemistry studies how to make such computer simulations possible by allowing certain approximations. This increasingly important area is driven by research in biomedical and technological applications such as drug design, DNA studies, material and chemical industries. Nowadays, a huge variety of molecular systems and chemical processes can be simulated on a computer. Efficient methods and usage of distributed-memory systems with many computational nodes enable large scale calculations, i.e. calculations on larger and more complex systems than ever before.

The electron is very light compared to the nuclei of atoms, and the motion of nuclei is typically much slower than the changes in the electronic state. In the Born-Oppenheimer approximation, it is assumed that electrons are moving in an electrostatic field generated by fixed nuclei. Electronic structure calculations are then focused on computing the structure of the electronic cloud, i.e. the spatial distribution of electrons in an atom or a molecule. Here I am concerned with the first principles, or ab initio, electronic structure methods, which use only the fundamental physical constants and do not have any parameters extracted from experimental data. Such calculations provide a great prediction quality, and they are not restricted to any specific group of systems.

A large effort in the development of modern computational methods is devoted to the computation of the electronic state with the lowest energy, referred to as the ground state. The electron density is the probability of finding an electron in a specific location. The ground state electron density is a fundamental quantity used to explain various chemical phenomena and describe properties of the molecular system. In the Hartree–Fock method [1] and Kohn–Sham density functional theory [2, 3], the ground state electron density is represented by the density matrix. The traditional diagonalization method for the
density matrix computation is restricted to rather small molecular systems due to its unfavorable cubic scaling behavior. It means that with doubling of the system size the execution time and the required memory are expected to increase eight times. Linearly scaling methods, which are the primary focus of this work, enable computational time and memory usage scaling proportionally to the number of atoms in the system. For an illustration, in Figure 1.1, I compare the execution times of two methods used for calculation of the density matrix, the linearly scaling recursive expansion method and the cubically scaling diagonalization method. More details are given in Chapter 2.

![Figure 1.1](image)

*Figure 1.1. Comparison of the execution times obtained for a cubically scaling method (diagonalization) and a linearly scaling method (recursive polynomial expansion) for computing the density matrix. Dashed lines show cubic and linear trend lines. Calculations are performed on the last SCF cycle of HF/3-21G calculations on glutamic acid and alanine helix systems using Ergo.*

**Thesis outline**

In the remainder of this chapter, I review the self-consistent field procedure for computation of the electronic ground state. In addition, section 1.3 addresses the decay properties of magnitude of matrix elements often encountered in electronic structure calculations. Chapter 2 begins with an introduction of methods for the computation of an inverse factor of the overlap basis set matrix. The inverse factor is used to formulate the density matrix problem in an orthogonal basis set, often allowing for simpler implementation and analysis. Then, various modern methods for the density matrix construction, in particular recursive polynomial expansions, are discussed. I introduce parameterless stopping criteria for recursive polynomial expansions in section 2.4. In section 2.5, the interior eigenvalue problem for computing eigenvalues and
eigenvectors located around the so called homo-lumo gap of the Fock/Kohn-Sham matrix is considered. I begin Chapter 3 by giving a brief introduction to the topic of parallel computing and reviewing modern parallel programming models. Then, I focus on the efficient computation of the density matrix in parallel on computer clusters. In the last chapter, I provide the thesis contributions together with a short summary of the included papers.

1.1 Electronic Schrödinger equation

Under the Born-Oppenheimer approximation, the behavior of matter at a subatomic scale can be described by the time-independent electronic Schrödinger equation

\[
\hat{H} \Phi_\alpha = E_\alpha \Phi_\alpha ,
\]

(1.1)

where \( \hat{H} \) is a Hamiltonian operator describing the motion of electrons in a field of fixed nuclei, \( \Phi_\alpha \) is a wavefunction describing a quantum state labeled by \( \alpha \), and \( E_\alpha \) is the corresponding energy level. For a system with \( N_e \) electrons, the wavefunction \( \Phi_\alpha = \Phi_\alpha (\vec{x}_1, \ldots, \vec{x}_{N_e}) \) depends on the electronic coordinates \( \vec{x}_1, \ldots, \vec{x}_{N_e} \), where each \( \vec{x}_i \) is a pair \( (\vec{r}_i, s_i) \) of the electron’s spatial and spin coordinates, respectively. The wavefunction itself does not have any direct physical meaning, but its square represents the probability of finding electrons in a given volume, referred to as the electron charge density:

\[
\rho_\alpha (\vec{r}) = N_e \int \cdots \int |\Phi_\alpha (\vec{x}_1, \vec{x}_2, \ldots, \vec{x}_{N_e})|^2 d\vec{x}_1 \vec{x}_2 \ldots \vec{x}_{N_e}.
\]

(1.2)

The scalar function \( \rho_\alpha \) depends only on three spatial coordinates \( \vec{r} \), and (formally) determines all properties of the system [2].

The quantum state \( \Phi_0 \), corresponding to the lowest energy level, is referred to as the ground state. It is the most stable state and all systems in nature tend to stay in it. Due to the large number of variables, the problem (1.1) can be solved exactly only for atoms or small molecules. Already for a single water molecule containing 10 electrons, the number of variables is 40. Thus, to deal with this dimensionality problem, methods introducing a set of approximations are required. In this work, I deal with two tools for ground state electronic density computation: the Hartree–Fock method [1] and Kohn–Sham density functional theory [2, 3]. Both methods require solution of an integro-differential equation, which, after a specific discretization, leads to a non-linear eigenvalue problem [4, 5]. A well established method for solving this eigenvalue problem is the self-consistent field procedure, which I introduce in the next section.
1.2 Self-consistent field procedure

In the Hartree–Fock (HF) method [1], the exact many-electron ground state wavefunction is approximated by a *Slater determinant* formed from \( N_e \) one-electron wavefunctions \( \phi_i(\vec{x}) \), which are unknown:

\[
\Phi_0(\vec{x}_1, \ldots, \vec{x}_{N_e}) \approx \frac{1}{\sqrt{N_e}} \begin{vmatrix}
\phi_1(\vec{x}_1) & \phi_2(\vec{x}_1) & \cdots & \phi_{N_e}(\vec{x}_1) \\
\vdots & \vdots & \ddots & \vdots \\
\phi_1(\vec{x}_{N_e}) & \phi_2(\vec{x}_{N_e}) & \cdots & \phi_{N_e}(\vec{x}_{N_e})
\end{vmatrix}.
\]

The functions \( \phi_i(\vec{x}) \) are often called molecular orbitals (MO). The HF method is a relatively computationally cheap method which gives a very rough approximation of \( \Phi_0 \). However, it is a powerful learning and theoretical tool, and the HF approximation of the ground state wavefunction may often be a good starting point for more advanced methods.

It can be shown that the Slater determinant (1.3) is an exact wavefunction of some fictitious system of non-interacting electrons [4]. The idea behind the Kohn–Sham density functional theory (KS-DFT) [2, 3] is to “change” the original complicated system of interacting electrons with an easier fictitious system of non-interacting electrons. Such auxiliary system has the ground-state wavefunction described by a Slater determinant, and, by construction, has the same ground state density as the original system. Having chosen the fictitious system, all remaining terms, neglected by this approximation, are incorporated into one functional. Unfortunately, the exact form of this functional is not known, but useful practical approximations are available, leading to many variations of KS-DFT.

In the HF approximation and KS-DFT fictitious system, the quantum state is represented using the Slater determinant formed by the one-electron wavefunctions \( \phi_i \). This allows to restate the Schrödinger equation as an eigenvalue problem with one-electron wavefunctions:

\[
\hat{f} \phi_i = \lambda_i \phi_i,
\]

where the one-electron Hermitian operator \( \hat{f} \) is referred to as *Fock or Kohn–Sham operator* in HF and KS-DFT, respectively.

The majority of compounds of interest in chemistry has an even number of electrons and they are referred to as closed-shell systems. One example of a closed-shell system is a water molecule containing 10 electrons. The calculations for such systems can be simplified by halving the set of MOs by considering only doubly-occupied MOs, where each MO \( \phi_i \) describes the motion of two electrons with different spins. Consequently, the wavefunctions depend only on the spatial coordinates, i.e. \( \phi_i = \phi_i(\vec{r}) \). Such calculations are referred to as *restricted*, and from now on I consider only this type of calculations.

Equations (1.4) are discretized using a given finite basis set \( \{ \psi_k \} \). The molecular orbitals are represented in this basis set as linear combinations of
the basis set functions:
\[ \tilde{\phi}_i = \sum_{k=1}^{N} c_{ik} \psi_k. \] (1.5)

In total, \( N \) molecular orbitals are formed from a basis set with \( N \) basis functions, but only \( N_{\text{occ}} = N_e/2 \) of them having the lowest energy will be occupied by a pair of electrons and form the Slater determinant. There is a large variety of basis sets used in practice [4, p. 97]. Common examples are atom-centered basis sets, which contain non-orthogonal one-particle functions centered on atoms. Often, atom-centered basis functions are chosen to be linear combinations of Gaussian functions.

The coefficients \( c_{ik} \), \( i,k = 1, \ldots, N \) become new unknown variables, and the equations (1.4) can be rewritten in the matrix form:
\[ \hat{F}(C)C = S \Lambda, \] (1.6)
where \( C \) is a coefficient matrix of size \( N \times N \) with elements \( c_{ik} \), \( \Lambda \) is a diagonal matrix \( \Lambda = \text{diag}(\lambda_1, \lambda_2, \ldots, \lambda_N) \) containing the eigenvalues arranged in ascending order \( \lambda_1 \leq \lambda_2 \leq \ldots \leq \lambda_N \). The matrix \( \hat{F} \) is referred to as the Fock or Kohn–Sham matrix in HF and KS-DFT, respectively, and it is defined as
\[ \hat{F}_{ij} = \int \psi_i^*(\vec{r}) \hat{f} \psi_j(\vec{r}) d\vec{r}. \] (1.7)

The basis set overlap matrix \( S \) is given by
\[ S_{ij} = \int \psi_i^*(\vec{r}) \psi_j(\vec{r}) d\vec{r}, \] (1.8)
and \( C^*SC = I \). I omit here the explicit expression for the elements of the matrix \( \hat{F} \) (see e.g. [5, p. 140]), but note that the elements of \( \hat{F} \) explicitly depend on the matrix of its eigenvectors \( C \).

In the restricted formulation at zero temperature, the ground state electron charge density (1.2) is equal to:
\[ \rho_0(\vec{r}) = 2 \sum_{i=1}^{N_{\text{occ}}} |\phi_i(\vec{r})|^2. \] (1.9)
Inserting (1.5) in (1.9) we get the representation of the electron density operator in the chosen basis set:
\[ \rho_0(\vec{r}) = \sum_{i,j=1}^{N} \hat{D}_{ij} \psi_i(\vec{r}) \psi_j^*(\vec{r}) \], (1.10)
where the matrix \( \hat{D} \) is called the density matrix and has the following elements:
\[ \hat{D}_{ij} = 2 \sum_{k=1}^{N_{\text{occ}}} c_{ik} c_{jk}^* = C \Delta C^*, \quad \Delta = \begin{bmatrix} 2I_{N_{\text{occ}}} & 0 \\ 0 & 0 \end{bmatrix}. \] (1.11)
Equation (1.6) is an eigenvector-dependent non-linear eigenvalue problem which is often solved using the self-consistent field (SCF) procedure. The Fock/Kohn–Sham matrix $\hat{F}$ can be represented using the density matrix elements. Thus, the iterative process for solving equation (1.6) can be built around the matrices $\hat{F}$ and $\hat{D}$ considering the density matrix $\hat{D}$ as the basic variable. The simplest form of such iterations is a fixed point iteration scheme:

$$\hat{D}_0 \rightarrow \hat{F}_1 \rightarrow \hat{D}_1 \rightarrow \hat{F}_2 \rightarrow \hat{D}_2 \rightarrow \ldots$$

Here the Fock/Kohn–Sham matrix $\hat{F}_1$ is obtained using the initial guess density matrix $\hat{D}_0$. Then, the eigenvalue problem (1.6) is solved with fixed matrix $\hat{F}_1$, and the new density matrix $\hat{D}_1$ is computed using (1.11). The stationary point is found when the density matrix $\hat{D}_i$ is the same as the density matrix from the previous SCF cycle $\hat{D}_{i-1}$. In practice, the fixed point iteration may diverge, requiring usage of some stabilization technique [6, p. 116].

In this work, I am focusing on the linear scaling SCF procedure, and the following section gives a theoretical justification of linear scaling methods.

1.3 Decay properties of matrix element magnitude

The linear scaling computational cost is achievable if matrices are sufficiently sparse and large. The matrix sparsity is maintained throughout the calculations by removal of small matrix elements, such that the number of non-zero matrix elements increases proportionally to the matrix size. This is possible due to the exponential decay of magnitude of matrix elements obtained by usage of the basis sets with bounded spatial extent of basis functions.

In Paper IV we study the localization properties of matrix sequences. To each square matrix $A_N$ of size $N$ we associate a pseudometric $d_N(i, j)$ defined on the index set $I_N = \{1\ldots N\}$ of $A_N$. The sequence of matrices $\{A_N\}_{N=1}^\infty$ satisfies the exponential decay property with respect to distance between indexes with constants $\alpha$ and $c$ if

$$|[A_N]_{ij}| \leq ce^{-\alpha d_N(i, j)}$$

for all $i, j = 1, \ldots, N$, and with $\alpha > 0$ and $c > 0$.

Let us consider a few examples of distance functions generating different decay patterns. The distance function $d_N^{\text{diag}}(i, j) = |i - j|$ generates exponential decay of the magnitude of matrix elements away from the diagonal. The distance function defined as $d_N^{\text{corner}}(i, j) = |N - i + j|$ generates decay from the left-bottom matrix corner. Figures 1.2a and 1.2b show elements of two matrices of size $N = 500$ with elements equal to $[A_N]_{ij} = e^{-0.1 d_N(i, j)}$ where $d_N = d_N^{\text{diag}}$ and $d_N = d_N^{\text{corner}}$, respectively.

In electronic structure calculations, each matrix element corresponds to a pair of basis functions, see e.g. (1.7) or (1.8). Then, the natural definition of
Figure 1.2. Illustration of exponential decay of magnitude of matrix elements generated by distance functions $d_N^{\text{diag}}$ (left panel) and $d_N^{\text{corner}}$ (right panel). Each point in a plot is equal to $\log([A_N]_{ij})$ where $[A_N]_{ij}$ is a corresponding matrix element.

The distance function $d_N(i, j)$ is the Euclidean distance between basis function centers corresponding to a matrix element $[A_N]_{ij}$.

In [7] the distance function is defined as a geodesic distance between two vertices in an undirected graph $G_N$, with $N$ vertices $I_{A_N} = \{1 \ldots N\}$ and $\mathcal{O}(N)$ edges, associated to matrix $A_N$ for each $N \geq 1$. The geodesic distance $g_N(i, j)$ between vertices $i$ and $j$ on a graph is equal to the length of the shortest path connecting them.

Note that the exponential decay property given by (1.12) does not guarantee that the number of negligible matrix elements grows proportionally to the system size $N$. In order to make a useful in practice definition, the set of allowed matrix sparsity patterns should be restricted. Let $|N_{d_N}(i, R)|$ be the number of indexes $j$ of the index set $I_{A_N}$ such that $d_N(i, j) < R$. Let $\nu > 0$ and $\beta > 0$ be constants independent of $N$. In Paper IV the restriction on allowed sparsity patterns is formally written as

$$|N_{d_N}(i, R)| \leq \nu R^\beta, \text{ for all } i \in I_{A_N}. \quad (1.13)$$

In [7] the restriction is imposed by assuming a bounded maximum degree (i.e. maximum number of neighbor vertices) of any node in graphs $G_N$ for all $N$. Then, for each vertex $i$, the number of vertices within distance 1 is bounded by a constant $\tilde{\nu} > 0$ independent on $N$, or in the notation of Paper IV:

$$|N_{g_N}(i, 2)| \leq \tilde{\nu}, \text{ for all } i \in I_{A_N}. \quad (1.14)$$

The condition (1.13) is more general in the sense that any pseudometric $d_N$ can be used. But if $d_N = g_N$, the condition (1.14) is less restrictive than (1.13) allowing a larger set of sparsity patterns.

In [7] it is shown that the number of non-negligible elements in each matrix row is bounded by a constant for all matrix sizes if conditions (1.12) with $d_N = g_N$ and (1.14) are satisfied. In Paper IV we show similar results in case
conditions (1.12) and (1.13) are satisfied with any pseudometric $d_N$. Thus, upon removal of small matrix elements, the number of non-zero matrix elements scales proportionally to the matrix size [8, 9, 10]. The computational cost of sparse matrix-matrix and sparse matrix-vector operations is proportional to the number of non-zero elements. Thus, if the sparsity is preserved throughout the calculations by removal of small matrix elements, the electronic structure methods may have computational time and memory usage scaling linearly with system size.
2. Density matrix construction

In the Hartree–Fock method (HF) [1] and Kohn–Sham density functional theory (KS-DFT) [2, 3], the Hamiltonian operator, in a given basis set, is represented by the Fock/Kohn–Sham matrix \( \hat{F} \), see Section 1.2. The self-consistent field (SCF) procedure is used to solve the non-linear eigenvalue problem (1.6), which I repeat here:

\[
\hat{F}(C)C = SC\Lambda. \tag{2.1}
\]

Let the eigenvalues in \( \Lambda \) be arranged in ascending order:

\[
\lambda_1 \leq \ldots \leq \lambda_{\text{Nocc}} < \lambda_{\text{Nocc}+1} \leq \ldots \leq \lambda_N, \tag{2.2}
\]

where I assume that \( \lambda_{\text{Nocc}} \neq \lambda_{\text{Nocc}+1} \), and \( \text{Nocc} \) is a given problem dependent value. The first \( \text{Nocc} \) eigenvalues correspond to occupied molecular orbitals and the rest correspond to unoccupied molecular orbitals. Of particular importance are \( \lambda_{\text{homo}} := \lambda_{\text{Nocc}} \), the eigenvalue corresponding to the highest occupied molecular orbital (homo), and \( \lambda_{\text{lumo}} := \lambda_{\text{Nocc}+1} \), the eigenvalue corresponding to the lowest unoccupied molecular orbital (lumo). Let \( \xi := \lambda_{\text{lumo}} - \lambda_{\text{homo}} > 0 \) be the homo-lumo gap. See Figure 2.1 for an illustration.

![Figure 2.1](image-url)

*Figure 2.1.* Illustration of the eigenvalue grouping for the Fock/Kohn–Sham matrix \( \hat{F} \), where “occupied” eigenvalues correspond to the molecular orbitals occupied by a pair of electrons and “unoccupied” eigenvalues correspond to the unoccupied molecular orbitals. The homo-lumo gap is assumed to be non-zero.

In each SCF cycle, the density matrix \( \hat{D} \) is built using (1.11) from the coefficient matrix \( C \), obtained by solving an eigenvalue problem (2.1) for a fixed matrix \( \hat{F} \). In general, basis sets used to discretize equations (1.4) are not orthogonal, i.e. \( S \neq I \). In an orthogonal basis set matrix expressions may be often
simplified requiring a smaller number of matrix operations. Assuming a fixed matrix $\hat{F}$, equation (2.1) rewritten in an orthogonal basis set is the ordinary eigenvalue problem

$$FY = Y\Lambda,$$  \hspace{1cm} (2.3)

where $Y = [y_1 \ y_2 \ \ldots \ y_N]$ is a set of orthogonal eigenvectors. The subspace spanned by the eigenvectors of the Fock/Kohn–Sham matrix $F$, that corresponds to the occupied orbitals, is called the *occupied invariant subspace* of $F$. The matrices $\hat{F}$ and $F$ in non-orthogonal and orthogonal basis sets, respectively, are connected using the congruence transformations:

$$F = Z^* \hat{F} Z, \quad ZY = C, \quad \text{and} \quad Z^* SZ = I.$$  \hspace{1cm} (2.4)

In the next section I provide more details about computation of the matrix $Z$ used in (2.4).

From equation (1.11) follows that if the basis set is orthogonal, then the scaled density matrix $D = \frac{1}{2} \hat{D}$ is equal to

$$D = Y \begin{bmatrix} I_{N_{\text{occ}}} & 0 \\ 0 & 0 \end{bmatrix} Y^*,$$  \hspace{1cm} (2.5)

and has the following properties:

$$D = D^* \quad \text{(Hermitian),}$$

$$\text{trace}(D) = N_{\text{occ}} \quad \text{(Given trace),}$$

$$D^2 = D \quad \text{(Idempotent).}$$  \hspace{1cm} (2.6)

From now on I will use the term density matrix to refer to the matrix $D$ given by (2.5).

### 2.1 Inverse factors of the basis set overlap matrix

Let $S$ be a Hermitian positive definite matrix. In this section, I consider the problem of computing a matrix $Z$ such that

$$S^{-1} = ZZ^*.$$  \hspace{1cm} (2.7)

The matrix $Z$ is referred to as an *inverse factor* of the matrix $S$. If the matrix $S$ is a basis set overlap matrix such that $S \neq I$, the density matrix $\hat{D}$ is computed from $\hat{F}$ in three steps:

1. congruence transformation $F = Z^* \hat{F} Z$;
2. density matrix $D$ construction from $F$;
3. congruence transformation $\hat{D} = 2 ZDZ^*$. 

18
In practice, the inverse factor $Z$ is computed only once at the beginning of the SCF procedure since the basis set does not change throughout the calculations.

To obtain a linear scaling execution time of the density construction procedure outlined above, the matrix $Z$ should be sparse. It has been shown in [7], that inverse factors, such as the inverse square root [11, 12, 13] and inverse Cholesky factor [14], possess the exponential decay property of matrix element magnitude with atomic separation. Similar results are obtained in Paper IV for the recursive [15] and localized (see Paper IV) inverse factorizations. Besides linear scaling execution time, another important concern is the efficient parallelization of the inverse factor algorithms. The hierarchical matrix representation [14] simplifies the implementation of sparse matrix operations in sequential and parallel environments. Thus, recursive algorithms, such as recursive [15] and localized (see Paper IV) inverse factorizations, utilizing hierarchical matrix representation are of great importance.

If an approximation $Q$ of the inverse factor such that $Q^*SQ \approx I$ is available, the iterative refinement procedure [16, 15] can be used to obtain the inverse factor of the matrix $S$

$$Z_{i+1} = Z_i \sum_{k=0}^{m} b_k \delta^k_i, \quad Z_0 = Q, \quad \delta_i = I - Z_i^*SZ_i,$$

(2.8)

where $b_0 = 1, b_k = \frac{2k-1}{2k} b_{k-1}, k = 1, \ldots, m$, and $m \geq 1$. The convergence of the iterative refinement (2.8) is guaranteed if the initial factorization error satisfies

$$\|\delta_0\|_2 < 1.$$  

(2.9)

Then, assuming $\|\delta_0\|_2 \neq 0$, it can be shown that the factorization error is systematically reduced:

$$\|\delta_{i+1}\|_2 < \|\delta_i\|_2^{m+1}.$$  

(2.10)

A binary principal submatrix decomposition of the matrix $S$ is given by

$$S = \begin{bmatrix} A & B \\ B^* & H \end{bmatrix},$$

(2.11)

where $A$ is a square matrix of size $1 \leq N_A < N$. Let $Z_A^*AZ_A = I$ and $Z_H^*HZ_H = I$ be inverse factorizations of $A$ and $H$, respectively. If the initial guess inverse factor is taken as:

$$Z_0 = \begin{bmatrix} Z_A & 0 \\ 0 & Z_H \end{bmatrix},$$

(2.12)

then the initial factorization error satisfies (2.9) ensuring the convergence of the iterative refinement [15]. To get inverse factors $Z_A$ and $Z_H$, matrices $A$ and $H$ are decomposed further in a similar way as $S$, and the iterative refinement
is applied to corresponding initial guesses. The subdivision continues recursively down to a single element or to a chosen submatrix size, for which an inverse factor can be easily computed.

Unfortunately, the memory usage and communication cost of the parallel recursive inverse factorization is high as shown in Paper IV. In Paper IV we reformulate the recursive inverse factorization algorithm and propose a novel localized inverse factorization. As I described in section 1.3, the non-negligible matrix elements in electronic structure calculations are localized, or concentrated, around certain regions. The proposed algorithm to a greater extent uses this locality property, and obtained operations on localized matrices can significantly reduce the communication cost in a parallel implementation (see Paper IV and [17]).

2.2 Overview of density matrix methods

The straightforward way to compute the density matrix (2.5) requires the diagonalization of the matrix $\mathbf{F}$. The diagonalization step scales cubically with matrix size, limiting calculations to rather small systems. Alternative approaches are usually based on matrix-matrix or matrix-vector multiplications (see e.g. [18, 19, 20]). If matrices are sufficiently large and sparse, the computational cost of such methods can be reduced to linear.

The polynomial expansion methods are based on the observation that the density matrix can be represented as

$$D = \theta(\mu I - \mathbf{F}) = Y \theta(\mu I - \Lambda) Y^*,$$  \hspace{1cm} (2.13)

$$\theta(x) = \begin{cases} 
0 & \text{if } x < 0, \\
1 & \text{otherwise},
\end{cases} \hspace{1cm} (2.14)$$

where $\mu$ is located inside the homo-lumo gap. The scaled and shifted Heaviside step function $\theta(\mu - x)$ is illustrated in Figure 2.2. Note that the step

![Figure 2.2. Illustration of the step function used for the density matrix representation in (2.13).](image)
function is viewed in (2.13) as a matrix function [21].

The non-smooth step function in equation (2.13) can be approximated by a polynomial. The approximation polynomial can be constructed recursively by composition of low degree polynomials \( p_i, i = 0, 1, 2, \ldots, n: \)

\[
\theta(\mu - x) \approx \beta_n(x) := p_n(p_{n-1}(\ldots p_0(x) \ldots)).
\]  

(2.15)

The initial polynomial \( p_0 \) is used to transform the eigenvalues of the matrix \( F \) into the required interval, often to \([0, 1]\), in reverse order. Then, polynomials \( p_i, i \geq 1 \) push eigenvalues that correspond to the occupied subspace to 1 and all other eigenvalues to 0. For this reason, the recursive polynomial expansion methods are often called purification methods. Note the rapid increase of the degree of the approximation polynomial with the number of iterations \( n: \) only by performing 10 matrix multiplications, the degree of the approximation polynomial can reach \( 2^{10} = 1024 \). The number of required matrix multiplications in the recursive polynomial expansion is proportional to the inverse homo-lumo gap \( \mathcal{O}(\log \frac{w}{\xi}) \) [22, 23], where \( w \) is the spectrum width \( w \) of the matrix \( F \).

An alternative to the recursive polynomial expansions is the approximation of \( \theta(\mu - x) \) by the Chebyshev polynomials [24, p. 234]. The convergence rate of the Chebyshev approximation is highly dependent on the differentiability of the approximated function [25, Theorem 5.14] [26], and the finite order Chebyshev approximations of non-smooth functions have a very peculiar behavior near discontinuities. For systems with non-vanishing homo-lumo gap, a function that is smooth near the step, can be used instead of the Heaviside step function. In general, such smooth functions are non-trivial to evaluate, and usually they are approximated using a Chebyshev polynomial expansion. The convergence speed of methods based on the Chebyshev approximation is in general small, so many matrix multiplications have to be performed to get satisfactory approximation quality. The degree of the Chebyshev approximation polynomial required to reach the requested approximation accuracy is proportional to the inverse gap \( \frac{w}{\xi} \) [27]. Thus, at least \( \mathcal{O}\left(\sqrt{\frac{w}{\xi}}\right) \) matrix multiplications is required to evaluate the polynomial [28].

Another widely used group of density matrix methods is based on the minimization of the functional given by:

\[
\Omega_0[D] = \text{trace}(DF),
\]  

(2.16)

where the minimization is performed with respect to all Hermitian matrices \( D \) in the presence of constraints on idempotency \( D = D^2 \) and trace (trace\( (D) = N_{occ} \)). The state-of-the-art unconstrained minimization problem, introduced by Li, Nunes and Vanderbilt [8], gives a good approximation of the solution to the original problem (2.16). In the new problem, the trace constraint is in-
corporated into the functional form and the idempotency constraint is relaxed:

\[ \tilde{\Omega}_0[D] = \text{trace}((3D^2 - 2D^3)(F - \mu I)). \]  \hspace{1cm} (2.17)

The McWeeny polynomial [29] \( p_{mw}(x) = 3x^2 - 2x^3 \), used in (2.17) and illustrated in Figure 2.3, moves a trial matrix \( D \) closer to idempotency as long as its eigenvalues are located in \( \left[ \frac{1 - \sqrt{3}}{2}, \frac{1 + \sqrt{3}}{2} \right] \) and there is no eigenvalue at 0.5. An alternative approach proposed in [9] does not require the knowledge of \( \mu \).

![Figure 2.3. The McWeeny polynomial.](image)

in advance. The minimization procedure in [9] preserves the trace of the initial guess density matrix by adjusting the value \( \mu \) in each minimization step. The minimization approaches using the conjugate gradient method converge linearly to the solution, whereas the recursive expansion methods have at least quadratic convergence when the matrix is close to idempotency [30, 23].

Some minimization methods involve parameterization of the density matrix and minimization of the corresponding energy functional. Given some density matrix \( D \), all other density matrices can be parameterized by elements of an anti-Hermitian matrix \( T \) [31]:

\[ f_D(T) = e^{-T}D e^T, \text{ where } T^* = -T. \]  \hspace{1cm} (2.18)

The iterations of the curvy-step minimization method [31, 32, 33] start with an initial guess density matrix \( D_0 \). In each iteration the matrix \( T_i \) is obtained by minimizing the energy functional

\[ \hat{\Omega}[T] = \text{trace}[e^{-T}D_i e^T F] \]  \hspace{1cm} (2.19)

with respect to the elements of the matrix \( T \), where the trial density matrix \( D_i \) is fixed. Then, the new density matrix approximation is obtained as \( D_{i+1} = f_{D_i}(T_i) \). The truncated Baker-Campbell-Hausdorff (BCH) expansion can be used to evaluate (2.18) [34, p. 81].
The appealing property of minimization approaches is that errors do not accumulate during the iterations. In the recursive and Chebyshev polynomial expansions, the accuracy loss due to numerical errors cannot be recovered in later iterations. However, error control schemes, connecting the forward error in the density matrix approximation with the truncation tolerance, have been developed for the recursive expansion methods [35]. Moreover, in Paper I we developed efficient parameterless stopping criteria for the recursive polynomial expansions. Our stopping criteria do not require any user defined input parameters and automatically adapt to changes of the truncation tolerance. To the best of my knowledge, there are no suggested methods to control the introduced numerical errors in the minimization methods or Chebyshev polynomial expansions. The stopping criteria tolerances, in Chebyshev polynomial expansion and minimization methods, are usually provided by the user.

2.3 Recursive polynomial expansions

A general form of recursive polynomial expansion algorithms is given in Algorithm 1. In this section, I provide a few instances of recursive polynomial expansions.

Algorithm 1 Recursive polynomial expansion (general form)

1: \( X_0 = p_0(F) \)
2: while stopping criterion not fulfilled, for \( i = 1, 2, \ldots \) do
3: \( X_i = p_i(X_{i-1}) \)
4: end while

The grand canonical purification, proposed by Palser and Manolopoulos [36], is based on the McWeeny polynomial \( p_{mw}(x) = 3x^2 - 2x^3 \), introduced in the previous section.

Recursive iterations \( X_{i+1} = p_{mw}(X_i), i \geq 0 \) converge quadratically when the matrix \( X_i \) is close to idempotency. The initial matrix \( X_0 \) is obtained by the following transformation of the matrix \( F \):

\[
X_0 = \frac{\mu I - F}{2 \max(\lambda_N - \mu, \mu - \lambda_1)} + 0.5I,
\]  

(2.20)

where the smallest \( \lambda_1 \) and largest \( \lambda_N \) eigenvalues of the matrix \( F \) can be estimated, for example, using the Gershgorin circle theorem or the Lanczos method. The transformation (2.20) ensures that the homo-lumo gap of \( X_0 \) overlaps 0.5.

The grand canonical purification is closely related to the matrix sign function methods for computing the density matrix [13]. The matrix sign-function of a given matrix \( A \) can be computed iteratively using the Newton-Schulz iter-


\begin{align*}
X_{k+1} &= p_{ns}(X_k) = \frac{1}{2}X_k(3I - X_k^2), \quad X_0 = A, \tag{2.21}
\end{align*}

where the spectrum of the matrix \( A \) is located in the interval \([-1, 1]\). The Newton-Schulz polynomial \( p_{ns} \) is just a scaled and shifted McWeeny polynomial \( p_{mw} \):

\begin{align*}
2p_{mw}\left(\frac{x+1}{2}\right) - 1 &= p_{ns}(x). \tag{2.22}
\end{align*}

The \textit{canonical purification} [36] conserves the trace throughout the iterations and does not require knowledge of the homo-lumo gap. The McWeeny polynomial has fixed points at 0, 1 and 0.5. In the new method the requirement that the polynomial has a fixed point at 0.5 is relaxed by modifying the McWeeny polynomial such that the fixed point can be anywhere in \([0, 1]\). Let

\begin{align*}
p^{(1)}_{pm}(x, c) &= \frac{(1+c)x^2 - x^3}{c}, \tag{2.23} \\
p^{(2)}_{pm}(x, c) &= \frac{(1-2c)x + (1+c)x^2 - x^3}{1-c}, \tag{2.24}
\end{align*}

and

\begin{align*}
p_{pm}(x, c) &= \begin{cases} 
p^{(1)}_{pm}(x, c) & \text{if } c \geq \frac{1}{2}, \\
p^{(2)}_{pm}(x, c) & \text{otherwise}, \tag{2.25}
\end{cases}
\end{align*}

see Figure 2.4 for an illustration. The fixed points of \( p_{pm}(x, c) \) in \([0, 1]\) are 0, 1 and \( c \). An initial transformation of \( F \) should ensure that the matrix \( X_0 \) has

![Figure 2.4](image-url)

Figure 2.4. Polynomial \( p_{pm}(x, c) \) given by (2.25) for various values of \( c \).
the correct trace and in each iteration $i > 0$ the parameter $c$ is chosen such that the trace is conserved. The canonical expansion gives poor performance for small and large ratio $\frac{N_{occ}}{N}$, since the slope of the polynomial $p_{pm}$ comes close to 1 either for the occupied or unoccupied eigenvalues.

The trace resetting method, proposed by Niklasson et al. [37], employs a linear combination of polynomials $R(x)$ and $G(x)$ with parameter $\gamma$

$$p_{trs}(x, \gamma) = R(x) + \gamma G(x).$$

As before, the initial transformation maps the eigenvalues of the original matrix to the interval $[0, 1]$. If $X_i$ is the matrix in iteration $i$, then the parameter $\gamma_i$ is chosen such that the matrix $X_{i+1} = p_{trs}(X_i, \gamma_i)$ has the correct trace. The parameter $\gamma_i$ is restricted to a certain interval of applicability $[\gamma_{min}, \gamma_{max}]$. If values of $\gamma_i$ outside this interval were allowed, eigenvalues of $X_i$ could be mapped outside the interval $[0, 1]$. Thus, when $\gamma_i$ is outside the interval of applicability, then one can reset the trace by using one or more iterations with the spectral projection polynomials of second degree (SP2) $p_{sp}^{(1)}(x) = x^2$ or $p_{sp}^{(2)}(x) = 2x - x^2$ such that:

$$
\begin{cases}
X_{i+1} = p_{sp}^{(1)}(X_i), & \text{if } \gamma_i < \gamma_{min}, \\
X_{i+1} = p_{sp}^{(2)}(X_i), & \text{if } \gamma_i > \gamma_{max}.
\end{cases}
$$

In [37], the authors propose to use the fourth order polynomials

$$R(x) = x^2(4x - 3x^2),$$
$$G(x) = x^2(1 - x)^2.$$  

Then, in (2.27), $\gamma_{min} = 0$ and $\gamma_{max} = 6$. The polynomial $p_{trs}(x, \gamma)$ with $R(x)$ and $G(x)$ given by (2.28) and (2.29), respectively, is shown in Figure 2.5 for various parameters $\gamma$.

The SP2 polynomials, used above in the trace resetting method, can be generalized to higher polynomial degrees:

$$p_b^{(1)}(x) = 1 - (1-x)^b(1+bx),$$
$$p_b^{(2)}(x) = x^b(1+b(1-x)),$$

for $b \geq 1$, see Figure 2.6. Niklasson [22] proposed trace correcting purification using asymmetric polynomials $p_b^{(1)}(x)$ and $p_b^{(2)}(x)$ for $b \neq 2$. In Algorithm 2 is given the trace correcting algorithm for $b > 2$. In the particular case $b = 1$ we get the SP2 polynomials and we should revert the inequality sign on Line 4. In case $b = 2$, both polynomials reduce to the McWeeny polynomial and Algorithm 2 cannot be applied. The polynomials $p_1^{(1)}$ and $p_1^{(2)}$ have fixed points just at 0 and 1. The polynomials $p_b^{(1)}$ and $p_b^{(2)}$ for $b > 2$ have an
additional fixed point in the interval $[0, 1]$, i.e. there is $d \in [0, 0.5]$ such that $p_b^{(1)}(d) = d$ and $p_b^{(2)}(1-d) = 1-d$. Therefore, a necessary condition for the recursive expansion convergence is a non-empty overlap of the homo-lumo gap with the interval $[d, 1-d]$, which is ensured by the initial transformation of the matrix $F$ on line 2. If polynomials with $b = 1$ are used, only two matrices need to be stored in memory during the whole recursive expansion. Moreover, only one matrix-matrix multiplication per iteration is required, which simplifies the efficient error control in the density matrix approximation [35].

**Figure 2.5.** Polynomial $p_{\text{trs}}(x, \gamma)$ for various parameters $\gamma$ where the polynomials $R(X)$ and $G(x)$ are given by (2.28) and (2.29) respectively.

**Figure 2.6.** Polynomials $p_b^{(1)}(x)$ and $p_b^{(2)}(x)$ given by (2.30) and (2.31) for various values of $b$.  

26
Algorithm 2 Trace-correcting method, $b > 2$

1: **input**: $F$, $\lambda_1$, $\lambda_N$, $d$
2: $X_0 = \frac{(1-2d)(\lambda_N I - F)}{\lambda_N - \lambda_1} + dI$
3: **while** stopping criterion not fulfilled, for $i = 1, 2, \ldots$ **do**
4: **if** trace$(X_{i-1}) > N_{\text{occ}}$ **then**
5: $X_i = p_b^{(1)}(X_{i-1})$
6: **else**
7: $X_i = p_b^{(2)}(X_{i-1})$
8: **end if**
9: **end while**
10: **output**: $i$, $X_i$

The condition on line 4 in Algorithm 2 is sensitive to the eigenvalue distribution. Let us consider a diagonal matrix of size 300, with 1 eigenvalue located at 0.49 and 299 eigenvalues located at 0.51. We set $N_{\text{occ}} = 1$ and apply the trace-correcting recursive expansion with $b = 1$. The movement of homo and lumo eigenvalues throughout iterations is presented in the left panel of Figure 2.7. The algorithm requires almost 30 iterations before the homo eigenvalue starts to converge to 1.

![Figure 2.7](image-url)

*Figure 2.7. Convergence of the trace-correcting method with $b = 1$ (left panel) and the SP2 recursive expansion given in Algorithm 3 (right panel). For both methods we do 50 iterations and present the location of homo and lumo eigenvalues in each iteration.*

If the estimates $\tilde{\lambda}_{\text{homo}}$ and $\tilde{\lambda}_{\text{lumo}}$ of the homo and lumo eigenvalues, respectively, are available [38], then they can be used for choosing polynomials in every iteration. The recursive expansion with SP2 polynomials, chosen based on the homo and lumo eigenvalue estimates, is presented in Algorithm 3. We apply Algorithm 3 to the diagonal test matrix described above and plot the
location of homo and lumo eigenvalues in each iteration in the right panel of Figure 2.7. Both homo and lumo eigenvalues show a rapid convergence to 1 and 0, respectively. The total number of iterations required to reach convergence is reduced almost twice compared with the original trace-correcting method. Note that this is an extreme test case, which can hardly be seen in practice. Nevertheless, by choosing polynomials based on the homo-lumo eigenvalue estimates, the dependence on the system size is removed.

Algorithm 3 SP2 recursive polynomial expansion with known homo-lumo estimates

1: **input:** $F$, $\tilde{\lambda}_{\text{homo}}$, $\tilde{\lambda}_{\text{lumo}}$, $\lambda_1$, $\lambda_N$
2: $X_0 = \frac{\lambda_N - \lambda_1 - F}{\lambda_N - \lambda_1}$
3: $l_0 = \frac{\lambda_N - \lambda_{\text{lumo}}}{\lambda_N - \lambda_1}$
4: $h_0 = 1 - \frac{\lambda_N - \lambda_{\text{homo}}}{\lambda_N - \lambda_1}$
5: **while** stopping criterion not fulfilled, for $i = 1, 2, \ldots$ **do**
6:  
7:  
8:  
9:  
10:  
11:  
12:  **end if**
13: **end while**
14: **output:** $i$, $X_i$

Recursive expansions can be based on rational functions instead of low-degree polynomials [39, 40]. Such schemes require the solution of a linear system with a Hermitian and positive definite matrix in every iteration. Schemes using rational functions can be applied at any temperature, but at zero temperature they are in general slower than the schemes discussed above.

**Accelerated recursive polynomial expansions**

The polynomials, used in regular recursive expansions discussed above, are monotonic in $[0, 1]$ and have fixed points at 0 and 1 and a certain number of vanishing derivatives at 0 and 1. Prior knowledge of homo and lumo eigenvalues allows for the design of non-monotonic projection polynomials, which accelerate convergence of the recursive expansion. Such polynomials may not have stable fixed points at 0 and 1.

Rubensson [41] proposed a scale-and-fold acceleration technique. I will illustrate the idea using the SP2 polynomials $x^2$ and $2x - x^2$ introduced above. If the eigenspectrum is stretched out outside the $[0, 1]$ interval below 0, the polynomial $x^2$ can be used to fold the eigenspectrum over itself and return it to
the interval $[0, 1]$. Similarly, if the eigenspectrum is stretched out outside the $[0, 1]$ interval above 1, the polynomial $2x - x^2$ can be used to fold the eigenspectrum back. Thus, we get polynomials $((1 - a) + ax)^2$ and $2ax - (ax)^2$ where $a \in [1, 2]$ defines the amount of scaling and folding. See Figure 2.8 for an illustration where the regular (SP2) and accelerated (SP2ACC) polynomials are compared. In practice, one chooses the scaling parameter $a$ based on the homo and lumo eigenvalue estimates and such that the occupied and unoccupied spectrum parts do not mix between each other, see for example Algorithms 4 and 5 in Paper I.

Suryanarayana [42] suggested optimized purification polynomials which give the largest possible reduction of the lumo eigenvalue and largest possible increase of the homo eigenvalue at every step. Such polynomials must be optimized in each iteration. The author shows that the optimized McWeeny polynomial has the same performance (in terms of convergence of eigenvalues and number of matrix multiplications) as the McWeeny polynomial accelerated by the scale-and-fold technique discussed above.

**Figure 2.8.** Regular and accelerated SP2 polynomials. The acceleration parameter $a = 1.5$.

### 2.4 Parameterless stopping criteria

Only one user-defined input parameter is needed for the computation of the density matrix in the quantum chemistry code Ergo (see Paper III): the maximum allowed error in the occupied invariant subspace. This is possible thanks to the automatic determination of the required truncation threshold values and parameterless stopping criteria developed in Paper I. To avoid unnecessary computations, one usually wants to stop the iterations when numerical errors
start to dominate and no more significant improvement in the result can be expected. If the stopping criterion is satisfied too early, the potential improvement of the result is missed. Moreover, such result may be obtained using larger truncation values for intermediate matrices at less cost. Often stopping criteria are in the form \textit{stop as soon as the error measure goes below the user defined tolerance}, putting a burden for deciding when to stop on the user. In addition, too tight tolerances may result in failure to stop at all. The stopping criteria proposed in Paper I are automatically adjusting to a requested accuracy and do not need any ad hoc threshold values.

Let $e_i$ be some error measure in iteration $i$ of an iterative algorithm. The sequence of errors $e_i$, $i = 0, 1, \ldots$ is converging with order $q$ if there exist constants $C > 0$ and $K > 0$ such that for all $i \geq K$ holds

$$e_i \leq Ce_{i-1}^q.$$  \hfill (2.32)

The convergence order can be computed numerically using

$$q_i := \frac{\log (e_i/C)}{\log e_{i-1}}.$$  \hfill (2.33)

The idea behind our stopping criteria in Paper I is to look at the numerically observed order of convergence $q_i$ and detect when it becomes smaller than the theoretical $q$. If the constant $K > 0$ is known, the stopping criterion can be formulated as

$$\text{stop as soon as } i \geq K \text{ and } e_i > Ce_{i-1}^q.$$  \hfill (2.34)

Alternatively, one can find the smallest value $C_q \geq C$ such that in exact arithmetic

$$r_i := \frac{\log (e_i/C_q)}{\log e_{i-1}} \geq q,$$  \hfill (2.35)

for all $i = 1, 2, \ldots$, and start to look for a drop in $r_i$ from the first iteration.

The common error measure of the density matrix approximation $X_i$ in iteration $i$ is the \textit{idempotency error}:

$$e_i = \|X_i - X_i^2\|_2.$$  \hfill (2.36)

The recursive expansion using SP2 polynomials $x^2$ and $2x - x^2$, introduced in section 2.3, can be divided into three phases, as illustrated in Figure 2.9 duplicated here from Paper I. The expansion is quadratically convergent only in the middle purification phase. For sequences $e_i$ and expansion polynomials $p_i$ in iteration $i$ of the SP2 recursive expansion we have that (see Paper I):

$$e_i \leq Ce_{i-2}^2 \text{ if } p_i \neq p_{i-1} \text{ and } i \geq K,$$  \hfill (2.37)
Figure 2.9. Illustration of the three phases for a recursive expansion of order $q = 2$ based on the SP2 polynomials. In the conditioning phase the matrix does not come closer to idempotency, but the condition number $\kappa$ for the problem of evaluating (2.13) is lowered. In the purification phase the condition number is close to 1 and idempotency is approached quadratically. In the stagnation phase numerical errors start to dominate and the matrix again does not come closer to idempotency. The upper panel shows what we call the observed orders of convergence $q_i$ and $r_i$. Throughout the conditioning and purification phases $r_i \geq 2$ but in the stagnation phase $r_i < 2$.

where the value $K \geq 2$ determines the start of the purification phase and is problem dependent. In Paper I we determine the value $C_q \leq C$ and start to look at the numerical convergence drop immediately.

Note that in (2.35) it may happen that $r_i = q$. This means that an infinitesimal deviation from $r_i$ may trigger the stopping criterion, which was indeed observed in our numerical experiments. We solved this problem by introducing an even larger value $\tilde{C}_q > C_q$ such that

$$e_i + \tau \leq \tilde{C}_q e_{i-2}^2$$

(2.38)

for a given deviation $\tau$ of the numerically observed idempotency error from the exact one in some iteration $i$. This means that the selected value $\tilde{C}_q$ sets a limit on how inaccurate calculations are allowed. In Paper I we derive lower and upper bounds for $\tilde{C}_q$ (given by inequalities (2.20) and (2.24) in Paper I) depending on $\tau$. We show both bounds obtained for the SP2 expansion in Figure 2.10. The lower bound guards that a small deviation from $r_i$ will not trigger the stopping criterion too early. If a too large value $\tilde{C}_q$ is chosen, the algorithm may not stop at all. Thus, the upper bound guarantees that the stopping criterion will be satisfied in the stagnation phase. In Paper I we selected
Bounds for $\tilde{C}_q$ depending on the deviation of the idempotency error $\tau$ for the stopping criterion in the SP2 recursive expansion. Any value in the shaded region may be chosen as $\tilde{C}_q$. The selected value sets a limit on how inaccurate calculations are allowed. The dashed line is equal to $C_q \approx 4.409$ which is derived in Paper I.

In Paper IV a stopping criterion in the form similar to (2.34) has been developed for the iterative refinement of the inverse factor of the basis set overlap matrix $S$ (see section 2.1 for more details). For a given value $m$, the factorization error $\delta_i = I - Z_i^*SZ_i$ satisfies the following inequality:

$$\|\delta_i\|_2 < \|\delta_{i-1}\|_2^{m+1}$$

for $0 < \|\delta_{i-1}\|_2 < 1$. \hspace{1cm} (2.39)

Then the stopping criterion can be written as

$$\text{stop as soon as } i \geq 1 \text{ and } \|\delta_i\|_2 \geq \|\delta_{i-1}\|_2^{m+1}.$$ \hspace{1cm} (2.40)

In contrast to the recursive expansion, it is unlikely that the stopping criterion will be triggered too early due to numerical errors. The reason is that the spectral norm of the factorization error is usually significantly smaller than 1 and the truncation errors are moderate.

2.5 Molecular orbitals around the homo-lumo gap

Molecular orbitals (1.5) describe the allowed spatial distributions of electrons in a molecule, see the illustration in Figure 2.11. Molecular orbitals are constructed using eigenvectors of the Fock/Kohn–Sham matrix $F$. As stated at the
beginning of this chapter, I am considering molecular systems with the non-zero homo-lumo gap. Molecular orbitals with energies around the homo-lumo gap provide insight into structural and physical properties of molecules, and affect the flow of many chemical reactions. Of particular importance are the homo and lumo orbitals themselves.

Figure 2.11. Molecular orbitals obtained in HF/3-21G calculations for the $C_{35}H_{34}O_4$ molecule (PubChem [43] CID: 11214529). Isosurfaces are created using the Gabedit program [44]. The isovalue is set to 0.015. Orbitals are sorted according to their corresponding energies (i.e. eigenvalues). The occupied orbitals are given as homo-$i$, where homo-0 is the homo itself. The unoccupied orbitals are given as lumo-$i$, where lumo-0 is the lumo itself.

If matrices are sufficiently large and sparse, recursive polynomial expansions produce the density matrix $D$ for a given matrix $F$ with linear scaling computation cost. However, compared to the traditional diagonalization approach, in recursive polynomial expansions individual eigenvectors of $F$ are not readily available. In Paper II our goal was to find an algorithm for the on-the-fly computation of homo and lumo orbitals during the recursive expansion with good accuracy, and linearly scaling time and memory usage with increasing system size.

In the shift-and-square method [45, 46], also known as the folded spectrum method, the matrix $F$ is transformed to $(F - \sigma I)^2$ for some selected shift $\sigma$. The matrix spectrum is folded over itself, and interior eigenvalues of $F$, located close to the shift, map to extremal eigenvalues of the transformed matrix, facilitating convergence of the Lanczos eigensolver. An important observation is that the separation of eigenvalues near the homo-lumo gap from the rest
of the spectrum is increasing during the conditioning phase (see Figure 2.9) of the SP2 recursive expansion. Let \( \beta_i(x) = p_i(p_{i-1}(\ldots p_0(x)\ldots)) \) be an approximation polynomial obtained in the recursive expansion iteration \( i \) and \( g_\sigma(x) = (x - \sigma)^2 \). Then, the polynomial \( g_\sigma(\beta_i(x)) \) can be used as a spectrum filter, separating the eigenvalues of interest from the rest of the spectrum [47]. See illustration of obtained polynomial filter in Figure 2.12, duplicated here from Paper II.

\[ \begin{align*}
\text{Figure 2.12. Illustration of the approximation polynomial } & \beta_{12}(x) \text{ and the polynomial filter } (\beta_{12}(x) - \sigma)^2 \text{ obtained in the purify-shift-and-square method in iteration 12 for a particular choice of } \sigma = 0.6 \text{ in the SP2 recursive expansion. The matrix } X_0 \text{ is a diagonal matrix with equidistant eigenvalues in the intervals } [0, 0.48] \text{ and } [0.52, 1]. \\
\text{The lumo eigenvalue is represented using a red star marker.} 
\end{align*} \]

To obtain a sharp eigenvalue filter the recursive expansion iteration \( i \) and shift \( \sigma \) in \( g_\sigma(\beta_i(x)) \) should be carefully selected. In Paper II we improve the original purify-shift-and-square method [47] by studying the movement of homo and lumo eigenvalues during the recursive expansion. For that purpose, we use homo and lumo eigenvalue estimates proposed in [38], which can be obtained at negligible cost. The selected shift and recursive expansion iteration ensure that the eigenvalue of interest is the extreme eigenvalue and that it has a good separation from the rest of the spectrum, facilitating the eigensolver convergence and implementation.

The shift-and-square transformation squeezes the eigenspectrum and mixes occupied and unoccupied states. The consequence is that the purify-shift-and-square algorithm, given in Paper II, enables efficient computation of only homo and lumo eigenvectors. However, for some applications, 20-30 eigenpairs around the homo-lumo gap are required. In this case, the purify-shift-and-project method proposed in [47] might be a good alternative to the full matrix diagonalization. In Paper VI we implement the purify-shift-and-project
method in Ergo and evaluate its performance. The number of iterations of the Lanczos eigensolver is directly related to the distribution of eigenvalues of interest. In Paper VI we show that, if the number of Lanczos iterations does not increase significantly with increasing system size, the computational time of the recursive expansion with the computation of multiple eigenvectors around the homo-lumo gap scales linearly.
3. Density matrix construction on distributed-memory computer clusters

Computer clusters are built up of many computers (called nodes) linked together through a fast local area network. Each node has its own memory not accessible directly by others. Communication is done by sending messages between nodes through the network. Message passing over the network is much slower than the local memory access. Thus, one should think about reducing communication as much as possible. On the other hand, each node should not stay idle for a long time, so preferably the work should be distributed equally between nodes during the whole program execution. The algorithm or computer program is scalable if the performance does not deteriorate with increasing problem size and the number of involved processing units.

3.1 Parallel programming models

Parallel programming models provide an abstraction of a computer system, suppressing complex details about the underlying hardware and memory layout from the application programmer. There is a huge number of parallel programming models which are suitable for a given computer architecture, and it is usually the suitability for the particular problem and/or personal preference that define the choice of a model.

The performance and usability of the programming model is defined by many factors. Data and work abstractions impose a set of restrictions on the low-level work and data representations. Ideally, introduced abstractions should help the application programmer to write efficient and more understandable source code. Work and data distribution over nodes may be defined statically when the program starts or dynamically by a runtime library during program execution. Load balancing techniques aim to optimize resource usage, reduce data access time, prevent memory overload and automatically adapt to a change in the number of available resources.

The current state-of-the-art tools for shared memory parallel programming are POSIX Threads [48], usually referred to as Pthreads, and OpenMP [49]. Both are considered very low-level programming models, which put most of the burden of identifying parallelism and data access synchronization on the programmer. In general, programs written using Pthreads and OpenMP are
very error prone, requiring a lot of care to avoid deadlocks, data races and performance degradation [50].

The message passing interface (MPI) [51] is a specification for message passing operations and the de facto standard for programming applications on distributed systems. The user does not need to specify explicitly the creation, scheduling and destruction of MPI processes. MPI provides an interface for the communication, but the user is responsible for assigning work to each process and making sure that the data are available when needed.

In the **global address space model** all processes have access to a global (virtual) shared memory space which may be distributed across many processors. Linda [52] and Concurrent Collections (CnC) [53] programming models are based on the concept of a virtual associative shared memory (VASM), referred to as tuple spaces in Linda and collections in CnC. Data stored in this memory has no associated virtual addresses and cannot be accessed by their physical addresses. Instead, data can be retrieved from the virtual shared memory only on the basis of their content by using template matching rules. In contrast to MPI, in VASM models the sender puts its data into the common shared space, but does not specify a receiver, and the receiver does not care who sent the message and how long time ago it happened.

In the **partitioned global address space (PGAS) model** the virtual shared memory is divided into mutually disjoint partitions. Different programming models define the address space partitioning and mapping of partitions into the physical locations. The application programmer is responsible for placing data into partitions. PGAS programs use the “single program multiple data” execution model. Example implementations of the PGAS model are Coarray Fortran [54], Multiphase shared arrays [55], Unified Parallel C [56], and Global Arrays [57].

The **asynchronous partitioned global address space (APGAS) model** extends the PGAS model with support for asynchronous tasks [58], which may be invoked locally and remotely, and each task may spawn dynamically other tasks. APGAS languages can manage irregular workloads and are suitable for heterogeneous machines. Example implementations of the APGAS model are X10 [59] and Chapel [60].

In the **Actor model** the program consists of a set of autonomous self-contained objects, called actors, which communicate only by sending and receiving messages. Actors are completely isolated from each other and never share any data. The actor’s state is changed only in response to messages. Thus, the model is not suitable for applications requiring fine-grained parallelism and low communication overhead. Popular implementations of the Actor model include the functional programming languages Erlang [61] and Elixir [62] and the object oriented language Charm++ [63].
In *task-based programming models* the work to be done is divided into smaller pieces called tasks, which are expressed by the user using work abstractions provided by the particular model. Such abstractions may be computational routines with attached descriptions of input and output data, as “codelets” in StarPU [64, 65] or “tasks” in Chunks and Tasks [66]. Data abstractions such as “chunks” in Chunks and Tasks [66] and “closures” in Cilk [67] may simplify the model’s design and facilitate efficient implementations. The runtime scheduler maps tasks into the available physical resources, usually according to some load balancing strategy. Work stealing [67] is one of the most popular load balancing techniques among implementations of task-based programming models, see for example Cilk [67], Chunks and Tasks [66], DuctTeip [68], SuperGlue [69], and StarPU [64]. In the physically distributed environment, the mapping of data into the available resources is often the user’s responsibility. However, Chunks and Tasks [66] move that burden to the runtime library. In the following section, I will consider the Chunks and Tasks programming model [66] with more details.

### 3.1.1 Chunks and Tasks programming model

In the Chunks and Tasks task-based programming model [66], the user splits work into smaller tasks and data into smaller “pieces” called chunks, and registers both to the library. The library is taking care of the distribution of chunks and tasks into the physical resources. A similar concept is introduced in Charm++ [63], where the data and work are encapsulated inside chare objects and the runtime library manages their distribution. However, in Charm++ chares communicate by explicit message passing, whereas in Chunks and Tasks all the communication is hidden from the user.

To each registered task and chunk is assigned a unique identifier, which is chosen by the library and which may store useful information about the chunk or task. For example, the chunk identifier may store the chunk location and its size. Since each task has input chunks, which were registered to the library, it will never happen that some task requests chunks with non-existent identifiers. The chunk type encapsulates the raw-data and it may store the identifiers of other chunks, allowing the creation of chunks hierarchies. As soon as chunks are registered to the library, they are read-only, similarly to Erlang [61]. This may introduce some inconvenience when just a small part of the chunk should be modified. Since chunks are read-only, instead of just modifying the existing chunk, a new chunk should be created and registered to the library. However, this restriction allows for the writing of programs which are free from data races.

Each task has a number of input chunks and only one output chunk. This restriction provides an equal view on the chunks and tasks identifiers. Thus, under the task registration, the user may provide as input parameters both
chunks and tasks identifiers. If more output data are required, the user may create a chunk hierarchy of output data. A task may read the content only of the input chunks and register new chunks and tasks. Tasks are scheduled for execution as soon as all input chunks are available. Thus the user does not need to specify explicitly the data or task dependencies, the execution order is determined dynamically during the runtime, based on the availability of input chunks.

The Chunks and Tasks programming model sets up an interface for expressing parallelism in an application software. It provides work and data abstractions and sets rules for the program structure. CHT-MPI [70] is a particular implementation of the Chunks and Tasks programming model. One process, called parent, starts the execution of the program, spawns the worker processes when the parallel region is encountered, and sends the requested task to a worker chosen at random. Then, the parent process waits for the result without interfering with further task execution. A simple code snippet, showing CHT-MPI basic usage, is given below:

```c
1 // code executed by the parent process
2 int main ()
3 {
4 ...
5     cht::start (); // create workers
6 ...
7     // get chunk identifier for data A
8     ChunkID cid_A = registerChunk (A);
9     // register task with name MyAmazingTask
10    ChunkID cid_t = executeMotherTask <MyAmazingTask>(cid_A);
11    ...
12     cht::stop (); // destroy workers
13 ...
14 }
```

The task scheduler in CHT-MPI is based on randomized work stealing [67]. As soon as a worker process runs out of work, it attempts to steal work from another worker chosen at random. Such adaptive parallelism facilitates the execution on heterogeneous systems, if the corresponding kernel implementations are provided.

Restrictions imposed by the Chunks and Tasks model guarantee determinism and freedom from deadlocks and data races. The user should expose the parallelism in terms of chunks and tasks hierarchies, and the library will dynamically distribute them over the available resources.

### 3.2 Chunks and Tasks matrix library

The Chunks and Tasks matrix library (CHTML), built upon the Chunks and Tasks programming model, was introduced in [71]. A matrix in CHTML is
represented using a hierarchy of chunk identifiers (ChunkIDs). The hierarchy is built based on a recursive disjoint matrix subdivision. In this work, I am using matrix division into four rectangular submatrices resulting in a quad-tree of ChunkIDs, but other subdivisions are possible. On the top level, the ChunkID corresponds to the whole matrix. The corresponding chunk stores ChunkIDs of four submatrices, establishing the parent-child relationships. Each child chunk stores four ChunkIDs of the further subdivision of the corresponding submatrix, and so on, until a requested submatrix size is reached. Matrix elements are stored only at the lowest (or leaf) level of the obtained ChunkID quad-tree. If along the way of building the quad-tree a submatrix is zero, the corresponding ChunkID is represented using the special CHUNK_ID_NULL identifier and it has no children. See Figure 3.1, representing the described ChunkID hierarchy for one example matrix.

![Figure 3.1. Illustration of the quad-tree chunk hierarchy for the matrix presented on the left panel. Blocks containing non-zero elements are displayed in blue color. Zero blocks, shown in white color in the matrix, are not stored. The corresponding quad-tree nodes are represented as empty circles.](image)

The leaf level matrix representation can be chosen based on the matrix type. If it is dense, one wants to store all matrix elements in the contiguous memory locations. If it is sparse, a sparse matrix storage format can be chosen based on the sparsity degree. In this work, I deal mostly with matrices which have thousands of non-zero elements per row, and the block-sparse matrix format [71] has been proven efficient for such cases.

I implemented a set of C++ wrapper classes which encapsulate general and symmetric matrix types and various matrix operations. The wrappers utilize the matrix chunk types and task implementations from CHTML, but provide a simplified interface. Each wrapper class stores two ChunkIDs. One ChunkID represents the matrix itself, which may be distributed between computer nodes. Another ChunkID refers to the matrix parameters, such as matrix dimensions and details about the leaf level matrix representation. If the wrapper class representing a symmetric matrix is used, then only the matrix elements above the main diagonal are stored. Usage of matrix symmetry allows to reduce the computational time and communication cost almost twice, see Paper V.
3.3 Parallel recursive polynomial expansion

In practice, larger problems require more computational resources. Thus, often the parallel efficiency is evaluated in weak scaling tests, where the problem size is increasing proportionally to the number of nodes/processes. In this case, the workload per process is kept constant. Ideally, the communication overhead per process should also remain constant regardless of the number of processes.

In [71] the weak scaling performance of the matrix-matrix multiplication in CHTML is evaluated using theoretical derivations and practical experiments. It is shown that, if the number of non-zero matrix elements increases proportionally to the number of processes, the amount of data sent and received per worker process is asymptotically constant. In Paper V we present the distributed-memory implementation of the SP2 and SP2ACC recursive polynomial expansions discussed in section 2.3, and analyze their parallel performance. Recursive expansions are implemented using wrapper classes described in the previous section. The accuracy of the density matrix approximation is controlled by the dynamic selection of threshold values throughout the iterations [35] and usage of parameterless stopping criteria described in section 2.4 and in Paper I. The same accuracy is requested for all systems of growing size. In Paper V we evaluate the weak scaling performance of the whole recursive expansion. We show that the execution time of recursive expansions scales as a squared logarithm of the number of processes, and the communication cost per worker process is asymptotically constant in a weak scaling limit.
4. Thesis contributions

In this chapter, I present the main contributions of my work and relate them to the corresponding scientific publications.

- **Parameterless stopping criteria** for density matrix recursive polynomial expansions are developed in Paper I. In Paper IV we derive the stopping criteria for the iterative refinement procedure, used for computing inverse factors of Hermitian positive definite matrices. In contrast to standard approaches, where the iterations are stopped when a given error measure goes below a user-defined threshold value, our stopping criteria do not require any ad hoc tolerances. The developed stopping criteria are based on the discrepancy between the theoretical and observed convergence orders at the stagnation phase of the algorithm. The efficiency of the stopping criteria is demonstrated by performing large scale calculations.

- A fast algorithm for the **computation of frontal orbitals** during recursive expansions is developed in Paper II. The explicit eigenvalues and eigenvectors of the Fock/Kohn-Sham matrix are not immediately available in linear scaling methods. The purify-shift-and-square method [47] for efficient computation of molecular orbitals during recursive expansions is based on two observations:
  1. a cheap matrix transformation can be used to modify the eigen-spectrum such that the desired internal eigenvalues map to extremal ones;
  2. the separation of the eigenvalues around the homo-lumo gap is increasing in the intermediate iterations of recursive expansions.
In Paper II we improve the original method from [47] and focus on computing only homo and lumo eigenvectors. We study the movement of eigenvalues during recursive expansions and carefully select the matrix transformations and recursive expansion iterations for computing homo and lumo eigenvectors. The computational cost is shown to be negligible compared to the cost of the whole recursive expansion.

- **Computation of multiple eigenvectors around the homo-lumo gap** using the purify-shift-and-project method, initially proposed in [47], is evaluated in Paper VI. We implement the purify-shift-and-project method in Ergo (Paper III) and compare its performance to other popular algorithms for computing multiple eigenvectors in linear scaling setting. If the eigenvalues of interest remain well separated from each other and the
rest of the spectrum for increasing system size, the number of iterations of the Lanczos eigensolver does not change significantly. Under this assumption, the execution time of recursive expansions with computation of multiple eigenvectors scales linearly.

- **Software development**: recursive expansions using polynomials $x^2$ and $2x - x^2$, and their accelerated versions [41], are implemented in the open-source quantum chemistry software Ergo, which is presented in Paper III. Current implementation includes the stopping criteria from Paper I, the method for the computation of homo and lumo orbitals from Paper II, and the method for computation of multiple eigenvalues and eigenvectors around the homo-lumo gap presented in Paper VI.

- **Parallelization**: the Chunks and Tasks matrix library [71] is extended and used for the parallelization of recursive expansions on distributed-memory systems. The weak scaling performance of parallelized recursive expansions is evaluated in Paper V.

- **Localized inverse factorization** for computing an inverse factor of a Hermitian positive definite matrix on distributed-memory clusters is proposed in Paper IV. An inverse factor of the basis set overlap matrix is used in density matrix computations. Formally, we present a reformulation of recursive inverse factorization developed in [15]. However, we show the superiority of the localized variant using our new theoretical tools for analysis of the matrix decay properties in electronic structure calculations. Moreover, we study the algorithms numerically using the Chunks and Tasks matrix library [17, 71] implementations, and show that the localized inverse factorization reduces drastically the communication cost.

Beteendet hos materia på subatomär skala kan beskrivas av den tidsobero-ende elektroniska Schrödingererekvationen

\[ \hat{H}\Phi_\alpha = E_\alpha \Phi_\alpha, \]

(5.1)
där \( \hat{H} \) är en Hamiltonoperator, \( \Phi_\alpha \) är en vågfunktion som beskriver ett kvant-tillstånd \( \alpha \), och \( E_\alpha \) är motsvarande energinivå. Mycket arbete i utvecklingen av moderna beräkningsmetoder ägnas åt beräkningen av elektron tillståndet \( \Phi_0 \) med lägst energi. Detta tillstånd kallas för grundtillstånd. Ett tillstånd definie-ras av hur elektroner är distribuerade runt en molekyl. En sådan fördelning kallas också elektronstruktur. Elektronstrukturen representeras av tät hetsma-trisen.

Detta arbete fokuserar på effektiv beräkning av tät hetsmatrisen \( D \) på distri-buerade datorsystem. Tät hetsmatrisen beräknas från en given matris \( F \) som re-prenterar en Hamiltonoperator \( \hat{H} \). Beräkningskostnaden för den metod som traditionellt används för att beräkna tät hetsmatrisen växer kubiskt med systemstorleken. Detta betyder att en dubblering av systemstorleken resulterar i att exekveringstiden åttafaldigas. Därför kan ett sådant tillvägagångssätt endast tillämpas på tämligen små system.

I elektronstrukturberäkningar är storleken på matriselementen exponenti-ellt avtagande. Detta kommer av en fysisk lokalitet för interaktioner. Om små matriselement tas bort från matrisrepresentationen så kan gleshet i matriserna bevaras igenom hela beräkningen. I detta arbete fokuserar jag speciellt på rekursiva polynomexpansionsmetoder. Idén är att välja polynom \( p_0, p_1, \ldots, p_n \) så att \( D \approx p_n(\ldots(p_1(p_0(F)))\ldots) \). Polynomens tillämpas iterativt tills något stoppvillkor är uppfyllt. På detta sätt uppnås en beräkningstid som växer endast linjärt med systemstorleken förutsatt att matriserna i den rekursiva expan-sionen är tillräckligt glesa.


De flesta av de metoder som diskuteras här är formulerade för ortogonala basset. En invers faktor till överlappsmatrisen behövs för transformation mellan ortogonalt och icke-ortogonalt basset. Den lokaliserade inversa faktoriseringen av överlappsmatrisen föreslagen i detta arbete har överlägsen parallell prestanda jämfört med andra metoder. Dessutom utvecklar vi nya teoretiska verktyg för analys av matriser med exponentiellt avtagande element.

6. Acknowledgments

Throughout my PhD studies I have received a lot of support and assistance. First of all, I would like to thank my supervisors Emanuel Rubensson and Elias Rudberg for their valuable guidance and encouragement, which helped me to go through the PhD smoothly. They taught me to think like a researcher, formulate my ideas clearly and ask right questions. In addition, I thank my colleague Anton Artemov for useful scientific discussions.

I thank Emanuel Rubensson and Giampaolo Mele for carefully reading my thesis and providing a valuable feedback. Moreover, I appreciate the help of Emanuel Rubensson in translating the Summary in Swedish.

I am extremely grateful to my friends and colleagues at the IT department, and particularly the division of Scientific Computing. It was a pleasure to work with you! The tradition of having coffee/tea (“fika”) together is really great. I will definitely miss it. I also wish to thank all “lunch-box” people for making our lunches really fun. Special thanks to my former office-mates, in particular Timofey Mukha, Saleh Rezaeiravesh, Diana Yamalova, and Tatiana Chistjakova. I also thank Igor Tominec for not letting me forget the Croatian language.

Many thanks to Grigorij Vasiljev for the help in making an image for my thesis cover.

The PhD journey would not have been possible without the support of my family. I would like to thank my mother for her unconditional love and continuous support. I thank my beloved husband Giampaolo Mele for his care, patience and emotional support. Finally, I thank my little son Björn, whose smile brings me joy and happiness beyond words.
References


A doctoral dissertation from the Faculty of Science and Technology, Uppsala University, is usually a summary of a number of papers. A few copies of the complete dissertation are kept at major Swedish research libraries, while the summary alone is distributed internationally through the series Digital Comprehensive Summaries of Uppsala Dissertations from the Faculty of Science and Technology. (Prior to January, 2005, the series was published under the title “Comprehensive Summaries of Uppsala Dissertations from the Faculty of Science and Technology”.)