TECHNISCHE
UNIVERSITÄT
WIEN

DIPLOMARBEIT

Spectrum Slicing method for eigenvector computation in Density Functional Theory

ausgeführt am

Institut für
Analysis und Scientific Computing
TU Wien

unter der Anleitung von

Prof. Dr. Joachim Schöberl, TU Wien
in Zusammenarbeit mit Dr. Marc Torrent bei CEA
Bruyères-le-Chatel, Frankreich

durch

Clémentine Barat
Matrikelnummer: 12122462

Wien, 10. Oktober 2023

Kurzfassung

Mit der Entwicklung hybrider Supercomputer, die aus mit GPU gekoppelten Recheneinheiten bestehen, steigt das Potenzial für Parallelität in wissenschaftlichen Codes erheblich. Die Programmiermodelle müssen an diese neue Computerhardware angepasst werden. Die Abinit-Software ist ein internationales Projekt, mit dem Materialeigenschaften auf mikroskopischer Ebene berechnet werden können. Dabei werden die Gleichungen der Quantenphysik für Elektronen und eine ebene-Wellen Basis verwendet. Ein iterativer Eigenwertlöser wird verwendet, um die Schrödinger-Gleichung zu lösen. Heutzutage ist der Löser auf modernen Supercomputern durch die Anwendung des sogenannten Rayleigh-Ritz-Verfahrens in seiner Leistung begrenzt.

In diesem Projekt erforschen wir die Möglichkeiten eines neuen Eigenwertlösers, bekannt als *Spectrum Slicing* Eigenlöser, der die mit dem Rayleigh-Ritz-Verfahrens verbundene Rechenlast reduzieren und parallelisieren kann. Es beruht auf der Aufteilung des Eigenwertspektrums in *slices*, die jeweils von einem anderen parallelen Prozess verarbeitet werden. In dieser Arbeit untersuchen wir diese Methode im Detail, mathematisch und mit einem Prototyp-Code in der programmierungssprache Julia erstellt. Wir zeigen, dass der *Spectrum Slicing* Löser tatsächlich große DFT-Berechnungen beschleunigen kann, verglichen mit den bereits in Abinit implementierten Eigenwert-Solvern.

Abstract

With the development of hybrid supercomputers, made of computational units coupled with GPU accelerators, the potential for parallelism in scientific codes is increasing significantly. The programming models have to be adapted to these new computing hardware. The Abinit software is an international project that allows to compute material properties at microscopic scale, using the quantum physics equations for electrons and a plane wave basis. An iterative eigenvalue solver is used to solve the Schrödinger equation. Nowadays on modern supercomputers, the solver is limited in performance by the application of the so-called Rayleigh-Ritz procedure.

In this project we explore the capabilities of a new eigenvalue solver, known as *Spectrum Slicing* eigensolver, which potentially reduces and parallelizes the computational load associated with the Rayleigh-Ritz procedure. It relies on splitting the eigenvalue spectrum into slices, each processed by a different parallel process. In this work, we examine this method in detail, mathematically and with a prototype code built in Julia language. We show that the Spectrum Slicing solver can indeed speed up large DFT calculations, compared with the existing eigensolvers already implemented in Abinit.

Acknowledgement

I want to express my deepest gratitude to Marc Torrent, who supervised my internship at the *CEA*, during which this master thesis was written. He was a great help in familiarizing me with the subject of DFT. Without his advice and the many discussions we had, this work would not have been possible.

I would also like to thank the *CEA* for allowing me to carry out my internship with them and write my master's thesis in a pleasant and stimulating environment, as well as my colleagues who make this environment pleasant and stimulating.

Special thanks to Antoine Levitt, who gave us his time for several fruitful discussions.

I would also like to thank my supervisor Joachim Schöberl, who was always available to help me.

Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die vorliegende Diplomarbeit selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt bzw. die wörtlich oder sinngemäß entnommenen Stellen als solche kenntlich gemacht habe.

Wien, am 10. Oktober 2023

Clémentine Barat

Contents

1. Introduction

1.1. Overview

This work was carried out during my internship at the CEA, a french public research institution.

Created in 1945 to develop the applications resulting from the atomic sciences, the French Atomic Energy and Alternative Energies Commission (CEA) is the instrument with which the french public research has equipped itself to conduct research involving major strategic and societal challenges. As part of this mission, the CEA develops new scientific knowledge and transfers technological innovations to the industrial world. The historic mission entrusted to the CEA has irrigated several fields of research directly or indirectly linked to the atomic sciences. Atomic research remains strategic for applications in the field of new technologies, low-carbon energies, digital technology, defense and health.

For its research needs and defense mission, the CEA co-develops supercomputers with the french supercomputer manufacturer Atos, at the best level in the world. In particular, the CEA operates the computers of the *Très grand centre de calcul du CEA* (TGCC) for the benefit of industry and research. The three largest supercomputers of the TGCC, ranked in the *Top500* (list of the fastest 500 supercomputers in the world), are *CEA HF* with a peak performance of 30 PFlop/s, *Joliot-Curie* with 15 PFlop/s and *Topaze* with 5 PFlop/s. These supercomputers need to be constantly updated to remain at the cutting edge of technology. The codes that run on these supercomputers must also evolve to remain adapted to the computer hardware and achieve the best performance improvements. Nowadays supercomputers are transitioning, with the addition of GPU-type accelerators in the computing units, adding many new possibilities of parallel computing.

The laboratory I was welcomed into uses these supercomputers to work on the properties of materials at microscopic level, in particular their electronic properties. The calculation are made from first principles approaches, meaning that they are only based on the first principles of quantum physics, with no experimental inputs. Notably, the researchers of this laboratory study the equations of state of a large variety of materials like metals, hydrides or alloys to determine their phase changes, stability under specific thermodynamic conditions,

etc. A lot of these studies are made in extreme conditions of temperature or pressure. Most of the computations are made using the Density Functional Theory (DFT) formalism with the Abinit software in which the laboratory is one the main developer group.

The Abinit software ([GAA+20], [Gro23]) is a package whose main program allows one to find the total energy, charge density and electronic structure of systems made of electrons and nuclei (molecules and periodic solids) within Density Functional Theory (DFT), using pseudopotentials (or PAW atomic data) and a planewave basis. This is an international collaborative project, under an open-source licence. The main program consists essentially in solving the Schrödinger equation, expressed as an eigenvalue problem. Because we are using a very large plane wave basis (which make the construction of a matrix unfeasible) and because we only need the lowest eigenvalues, the algorithms used are very specific (e.g. iterative eigensolvers) and need to be continually adapted to the supercomputers.

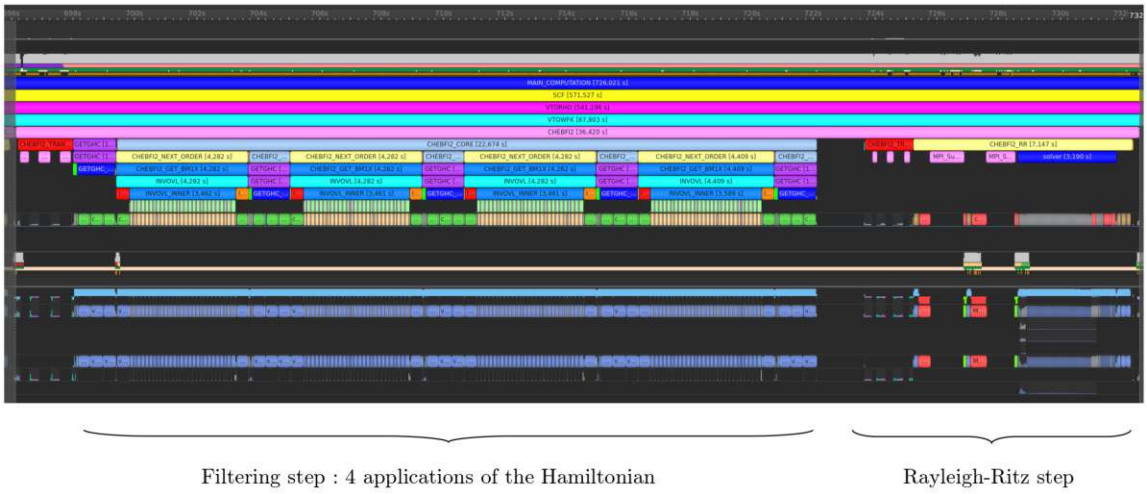
The aim of this work is to study a new iterative "matrix free" eigensolver, which could be very well suited to CPU+GPU computers because of its high potential for parallelism. As of today, there are two parallel eigensolvers implemented in Abinit, LOBPCG (Locally Optimal Block Preconditioned Conjugate Gradient method) and ChebFi (Chebyshev Filtering), which will be described below. Both algorithms use, as part of their process, the Rayleigh-Ritz method which turns out to be the blocking factor on modern super computers, as illustrated in figure 1.1, because of its lack of parallelizability. As illustrated by the *Amdahl law*, increasing computational resources above a certain threshold won't speedup the computation because of the inevitable non-scalable step that is Rayleigh-Ritz. Observing figure 1.1, we see that we are close to this threshold with the new GPU-based computers.

In this work we study the *Spectrum Slicing* method that aims at reducing the time spent in the Rayleigh-Ritz step, by parallelizing the code and applying the Rayleigh-Ritz on a smaller subspace. We specifically study its applications in plane-wave DFT, involving dense matrices, and estimate its potential efficiency compared to the already implemented iterative eigensolvers.

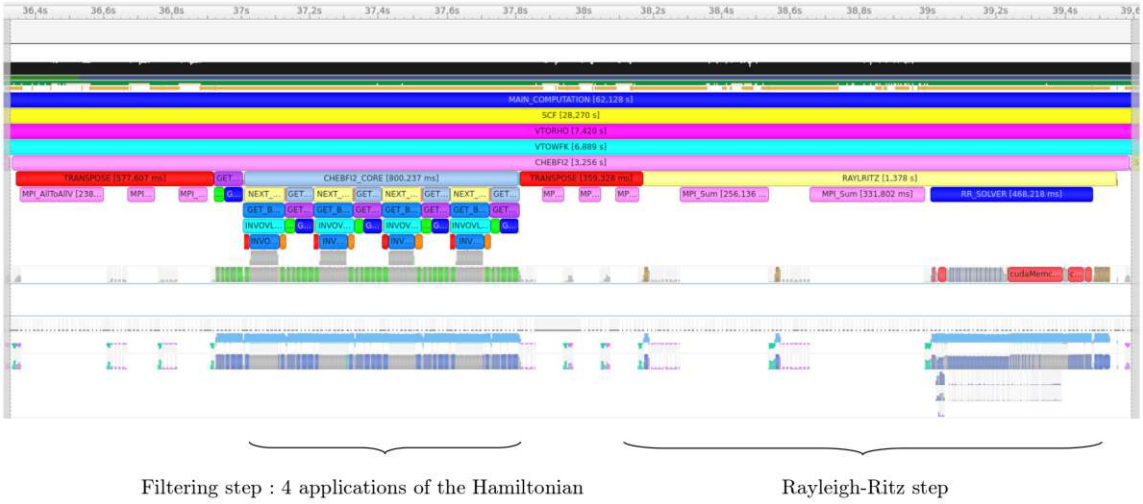
1.2. Thesis structure

The thesis will be organised as follows.

Chapter 2 is devoted to the Density functional theory formalism. We first introduce the physical problem that we are interested in, whose formulation is known but impossible to solve in practice. Then we present the ground idea of Hohenberg and Kohn that support DFT and reformulate the problem in lower dimension but with an unknown mathematical



(a) CPU



(b) CPU+GPU

Figure 1.1.: Time spent in the filtering step and in the Rayleigh-Ritz step during one SCF iteration with the Chebyshev filtering method.

expression. We present the *Ansatz* of Kohn and Sham allowing us to reformulate the problem as a non linear eigenvector problem, with a known expression, that can be realistically solved. In the last two sections, we focus on the numerical resolution of the Kohn-Sham's equations, by presenting first their discretization in a plane wave basis and then the self-consistent field (SCF) method used to solve the equations. This method calls for iterative eigensolvers, that are the main topic of this work.

In chapter 3, we present iterative eigensolver algorithms on Hermitian operators. The first section is dedicated to the Rayleigh-Ritz procedure that allow one to retrieve an estimation of eigenvectors from a subspace estimation. It is used in all the eigensolver algorithms involved in this study. Iterative eigensolvers are presented in two categories. Firstly, we introduce the algorithms used to solve the problem written in the form of an optimization problem, which are the most commonly used algorithms in Abinit today (e.g. LOBPCG). Secondly, we present algorithms acting on a subspace with filters (e.g. Chebyshev Filtering). These type of algorithms show better scalability. The *Spectrum Slicing* method belongs to the second category.

Chapter 4 adresses the *Spectrum Slicing* method in detail. It is organised in three parts, covering the three crucial points of the method: the choice of slices, the choice of filters and the merging of results. The idea of Spectrum Slicing is to split the eigenvalue spectrum of the operator to diagonalize in slices and look for the eigenvectors corresponding to each slice on different process in parallel. Choosing appropriate spectrum slices is essential to ensure fast convergence and in the case of SCF calculations, we have to take into account that the matrix to diagonalize is changing. Building the filters needed for Spectrum Slicing is more challenging than for Chebyshev Filtering. Here we introduce two types of filters, polynomial filters and rational filters.Finally we present the techniques used to merge the results from each slices. This last step is not the most challenging but is crucial to ensure correct results and convergence in SCF calculations.

Lastly in Chapter 5 we compare the time efficiency of Spectrum Slicing with Chebyshev filtering. As this work focuses mainly on the mathematical aspect of the method, the comparison is made analytically using a simplified modeling presented in the first section. Then we estimate the time complexity of the different steps needed in both methods and measure time on actual computation to estimate the time speedup. We can compare the estimated computation times in both method and make a projection of the system size at which Spectrum Slicing becomes cost-effective compared to Chebyshev Filtering. We compare it on some examples of physical systems of interest.

Finally, in the appendix, we recall some theory about Chebyshev Polynomials and Chebyshev series expansions (A), which are used all throughout this work. In a second appendix, we provide some parts of the *Julia* code used to implement Spectrum Slicing method (B).

2. Density functional theory

The density functional theory (DFT) is a computational modelling method used to solve quantum systems. It has proven to be especially useful to study the electronic structure of atomic systems, which are an example of many body systems. The theory rests on the two theorems of Hohenberg and Kohn from 1964 [HK64]. It is widely use today in Physics as well as Chemistry.

In this section we will present the theory behind DFT and more precisely behind Kohn-Sham's DFT to get the Kohn-Sham system, expressed as a non-linear eigenvector problem and we will present its numerical resolution using iterative eigensolvers.

2.1. The Schrödinger equation for a system of electrons

In quantum mechanic, the state of a stationary system is represented by its wave function Ψ, that must be a solution to the Schrödinger equation

$$H\Psi = E\Psi. \tag{2.1}$$

The operator H, acting on the wave function space, is called the *Hamiltonian*, its formulation depends on the system considered and E is the total energy of the system (scalar). This is an eigenvalue problem in the space of wave functions, and we usually look for the *ground state* of the system which is the unique solution with smallest energy E_0.

Here we consider a collection of m steady-state electrons under the influence of an external potential and the mutual Coulomb repulsion. The corresponding Hamiltonian is given by

$$H = T + V + U \tag{2.2}$$

where T, V and U are respectively the kinetic energy potential, external potential energy and Coulombic interaction operators. For this system, the wave functions depend on the

spin and position of each electron, that is

$$\begin{array}{rccc} \Psi: & (\mathbb{R}^3 \times \{-\frac{1}{2}, \frac{1}{2}\})^m & \to & \mathbb{C} \\ & (x_1, \ldots, x_m) & \mapsto & \Psi(x_1, \ldots, x_m) \end{array}$$

with $x_i = (r_i, \sigma_i)$ for $i = 1, \ldots, m$. The module of Ψ can be seen as the probability that the electrons are at positions $\{r_i\}_i$ and spins $\{\sigma_i\}_i$. Then, the expressions of the operators T, U and V (in atomic units) are

$$T\Psi(x_1, \ldots, x_m) = -\sum_{i=1}^{m} \frac{1}{2} \nabla_{r_i}^2 \Psi(x_1, \ldots, x_m) \tag{2.3}$$

$$V\Psi(x_1, \ldots, x_m) = \sum_{i=1}^{m} v(r_i) \Psi(x_1, \ldots, x_m) \tag{2.4}$$

$$U\Psi(x_1, \ldots, x_m) = \sum_{i=1}^{m} \sum_{j=i+1}^{m} \frac{1}{|r_i - r_j|} \Psi(x_1, \ldots, x_m) \tag{2.5}$$

where $v(r)$ is the external potential. If we consider a collection of atoms with fixed positions, $v(r)$ is the Coulombic interaction with the nucleus, that is

$$v(r) = \sum_{j=1}^{M} \frac{Z_j}{|R_j - r|} \tag{2.6}$$

where M is the number of nuclei, $Z_1, \ldots, Z_M$ are their atomic numbers and $R_1, \ldots, R_M$ their positions.

Except for a some very simple cases such as the Hydrogen atom ($m = 1$), this system is not possible to solve exactly, because of its very high dimension. The idea of DFT is to get round this problem of dimension, by expressing the problem not in terms of the wave function but in terms of the electronic density, which is only a function of the 3-dimensional real space. This is made possible by the theorems of Hohenberg and Kohn [HK64].

2.2. Hohenberg and Kohn theorems

The electronic density of the state described by the wave function Ψ is defined as

$$\begin{aligned}\rho(r) &= \langle \Psi, \textstyle\sum_{i=1}^{m} \delta(r - r_i)\Psi \rangle \\ &= \int d\sigma_1 \dots d\sigma_m \int d^3r_2 \cdots \int d^3r_m \bar{\Psi}((r,\sigma_1), x_2, \dots, x_m)\Psi((r,\sigma_1), x_2, \dots, x_m) \\ &+ \int d\sigma_1 \dots d\sigma_m \int d^3r_1 \int d^3r_3 \cdots \int d^3r_m \bar{\Psi}(x_1, (r,\sigma_2), x_3, \dots, x_m)\Psi(x_1, (r,\sigma_2), x_3, \dots, x_m) \\ &+ \dots \\ &+ \int d\sigma_1 \dots d\sigma_m \int d^3r_1 \cdots \int d^3r_{m-1} \bar{\Psi}(x_1, \dots, (r,\sigma_m))\Psi(x_1, \dots, (r,\sigma_m)).\end{aligned}$$

It describes the probability density of finding an electron at position r. Since electrons are indistinguishable (these are *fermions*), ρ can be rewritten

$$\rho(r) = m \int d\sigma_1 \dots d\sigma_m \int d^3r_2 \cdots \int d^3r_m \bar{\Psi}((r,\sigma_1), x_2, \dots, x_m)\Psi((r,\sigma_1), x_2, \dots, x_m).$$

At the ground state, the wave function Ψ is uniquely determined by the Hamiltonian operator, and consequently, by the external potential. Therefore the density ρ is a functional of the external potential v, meaning that there is a mapping

$$\begin{aligned}\mathbb{C}^{(\mathbb{R}^3)} &\to \mathbb{C}^{(\mathbb{R}^3)} \\ v &\mapsto \rho\end{aligned}$$

that associate potentials to their corresponding density. This mapping is called a functional because it is between function spaces. From here, we will denote functional with their parameter function in brackets e.g. $\rho[v]$. The first theorem of Hohenberg and Kohn states that the reverse is also true, i.e. the potential is a unique functional of the electronic density.

Theorem 2.2.1 (Hohenberg and Kohn 1964)**.** *The external potential (and hence the total energy) is a unique functional of the electron density.*

The proof, that is omitted here, is made by contradiction assuming that two different potentials can yield the same electronic density. Then, the energy functional $E[\rho]$ can be expressed as

$$E[\rho] = F[\rho] + E_V[\rho], \tag{2.7}$$

where $E_V[\rho]$ is the external potential energy functional and $F[\rho] = E_T[\rho] + E_U[\rho]$ is the

universal Hohenberg and Kohn functional, containing the kinetic energy and Coulombic interaction energy, that does not depend on the external potential.

Theorem 2.2.2 (Hohenberg and Kohn 1964)**.** *For a given potential v and number of atoms m, the functional*

$$E_{v,m}[\rho] = F[\rho] + \int v(r)\rho(r)dr \tag{2.8}$$

reaches its unique minimum on the densities ρ such that $\int \rho(r)dr = m$ at the ground state density.

Proof. Let ρ_0 the ground state density associated with this potential and number of atoms and Ψ_0 the corresponding wave function. By definition, the energy of the system is

$$E[\rho_0] = E_{v,m}[\rho_0] = \langle \Psi_0, V\Psi_0 \rangle + \langle \Psi_0, (T+U)\Psi_0 \rangle.$$

Now let ρ' a different density. The density ρ' is associated to a wave function Ψ' and describe the ground state of a potential v'. The energy is

$$\langle \Psi', V\Psi' \rangle + \langle \Psi', (T+U)\Psi' \rangle = \int v(r)\rho'(r)dr + F[\rho'] = E_{v,m}[\rho'],$$

since F is universal and valid for any potential. Now from the variational principle on wave functions, we have that $E_{v,m}[\rho'] > E_{v,m}[\rho_0]$. □

Thus, if the expression of $F[\rho]$ is known, our problem becomes a minimization problem on the set of acceptable densities, which are only 3 dimensional. Unfortunately, the expression of F is unknown and we will need to use some approximations, known as the equations of Kohn and Sham.

2.3. The Kohn-Sham density functional theory

In 1965, Kohn and Sham [KS65] hypothesized that there exist a system of m non interacting electrons that has the same electronic density as the system of m interacting electrons in the external potential $v(r)$. This assumption is known as the *Ansatz* of Kohn and Sham. In order for the two systems to have the same electronic density, we define a modified external potential $v_s(r)$ that reproduces the effect of the interactions between electrons. This means replacing the expression of the energy functional

$$E[\rho] = F[\rho] + E_V[\rho] = E_T[\rho] + E_U[\rho] + E_V[\rho]$$

with

$$E_s[\rho] = E_{T_s}[\rho] + E_{V_s}[\rho]$$

where T_s is the single particle kinetic energy potential. To ensure that the energy remains the same, we need $E_{V_s}[\rho] = E_V[\rho]+E_U[\rho]+(E_T[\rho]-E_{T_s}[\rho])$. At this point, no approximation has been made.

Now it remains to find an expression of the modified potential V_s. Kohn and Sham have expressed it using the Hartree potential and introducing a new term called exchange and correlation,

$$V_s[\rho] = V[\rho] + V_H[\rho] + V_{\text{xc}}[\rho]. \tag{2.9}$$

The Hartree potential is defined as

$$v_H(r) = \frac{1}{2}\int \frac{\rho(r)\rho(r')}{\|r-r'\|}dr' \tag{2.10}$$

and is an approximation of the Coulombic interaction term U. We note that the Hartree potential includes a self-interaction term that does not exist in U. The exchange and correlation term has an unknown expression that should compensate for all the approximations made, in the kinetic energy operator and Coulomb operator, that is

$$V_{\text{xc}}[\rho] = T[\rho] - T_s[\rho] + U[\rho] - V_H[\rho]. \tag{2.11}$$

Its expression with respect to the density is unknown. To run DFT computation, one needs an estimation of this term, many of which exist, but won't be developed here. This is where Kohn-Sham's DFT becomes an approximation.

One of the benefit of this method is that we now have to solve the Schrödinger equation for a system of non interacting electrons, which is much simpler. Indeed, for non interacting electrons, the ground state many-body wave function can be expressed as a Slater determinant

$$\Psi(x_1,\dots,x_m) = \frac{1}{\sqrt{m!}}\begin{vmatrix} \phi_1(x_1) & \phi_2(x_1) & \dots & \phi_m(x_1) \\ \phi_1(x_2) & \phi_2(x_2) & \dots & \phi_m(x_2) \\ \vdots & \vdots & & \vdots \\ \phi_1(x_m) & \phi_2(x_m) & \dots & \phi_m(x_m) \end{vmatrix}$$

where $\phi_1,\dots,\phi_N$ are called the Kohn-Sham orbitals and solve the one electron Schrödinger equation

$$(T_s + V_s)\phi_i(r,\sigma) = \left(-\frac{1}{2}\nabla_r^2 + v_s(r)\right)\phi_i(r,\sigma) = \epsilon_i\phi_i(r,\sigma)$$

for $\epsilon_1, \ldots, \epsilon_N$ the m smallest electronic energies (counted with multiplicity). The use of the Slater determinant ensure that Pauli's exclusion principle is satisfied. The corresponding density is

$$\rho(r) = \sum_{i=1}^{m} |\phi_i(r)|^2.$$

If we don't want to compute the ground state, but a higher energy one, we can use for instance Fermi-Dirac statistics and the density will be given by

$$\rho(r) = \sum_{i=1}^{m'} \bar{n}_i |\phi_i(r)|^2$$

where $\bar{n}_i$ are the occupations factor of orbital ϕ_i for $i = 1, \ldots, m'$ with $m' > m$. This way, we have replaced one high dimensional equation with a system of m (or m') smaller dimensional ones.

This yields the Kohn-Sham's equations in which we want to find the lower energy solutions $(\phi_1, \epsilon_1), \ldots, (\phi_{m'}, \epsilon_{m'})$ of

$$\begin{cases} \left(-\frac{1}{2}\nabla_r^2 + V + V_H[\rho] + V_{\text{xc}}[\rho]\right)\phi_i = \epsilon_i \phi_i \text{ for } i = 1 \ldots m' \\ \rho(r) = \sum_{i=1}^{m'} \bar{n}_i |\phi_i(r)|^2 \end{cases} . \tag{2.12}$$

2.4. Discretization of the problem

In order to solve this problem numerically, we need to discretize it. This means that we want to find a basis of functions $f_1, \ldots, f_N$ from $\mathbb{R}^3 \times \{-\frac{1}{2}, \frac{1}{2}\}$ to $\mathbb{C}$ in which any Kohn-Sham orbital ϕ can be expended as a linear combination:

$$\phi = \sum_{j=1}^{N} \alpha_j f_j.$$

Then ϕ is described by its coefficient vector in $\mathbb{C}^N$

$$\alpha = \begin{pmatrix} \alpha_1 \\ \vdots \\ \alpha_N \end{pmatrix}.$$

The Hamiltonian applied to ϕ, $H\phi$, is described by the matrix vector product $\bar{H}\alpha$ with

$$\bar{H} = \begin{pmatrix} \langle Hf_1, f_1\rangle & \dots & \langle Hf_1, f_N\rangle \\ \vdots & & \vdots \\ \langle Hf_N, f_1\rangle & \dots & \langle Hf_N, f_N\rangle \end{pmatrix}.$$

The matrix $\bar{H}$ is hermitian since the Hamiltonian operator is self-adjoint.

Thus, the Kohn-Sham equations can be rewritten in $\mathbb{C}^N$ as the non linear eigenvector problem

$$H(U)U = U\Lambda \tag{2.13}$$

for $U \in \mathbb{C}^{N\times m}$ and $\Lambda \in \mathbb{C}^{m\times m}$ a diagonal matrix with the m smallest eigenvalues of $H(U)$ counted with multiplicity. Here H is function from $\mathbb{C}^{N\times m}$ to $\mathbb{C}^{N\times N}$ that associate to U the hermitian matrix $H(U)$ corresponding to the density obtained from the columns of U.

Now obviously the initial Kohn-Sham's equation is in infinite dimension so it cannot be exactly discretized with a finite number of functions. But we can hope to get a good approximation with a well chosen basis.

In a code like Abinit, the discretization is done in a plane wave basis that is particularly suitable for periodic systems. We consider a periodic system whose unit cell is described by the Bravais lattice vectors $a_1, a_2, a_3 \in \mathbb{R}^3$. We know from Bloch's theorem [Blo29] that the Kohn-Sham's orbital can be written

$$\phi(r,\sigma) = e^{i\langle k,r\rangle}u(r,\sigma)$$

with $k \in \mathbb{R}^3$ and u that has the same periodicity as the system (for $\alpha, \beta \in \mathbb{R}^3$, $\langle\alpha,\beta\rangle$ denote their canonical inner product). Then u can be expended in Fourier series, that is

$$u(r,\sigma) = \sum_{j_1,j_2,j_3=-\infty}^{+\infty} c_{j_1,j_2,j_3} e^{i2\pi\langle j_1b_1+j_2b_2+j_3b_3,r\rangle}$$

for some c_{j_1,j_2,j_3} in $\mathbb{C}$ and where b_1, b_2, b_3 are the reciprocal lattice vectors, defined by

$$\langle b_i, a_j\rangle = \delta_{ij} \text{ for } i,j = 1,2,3.$$

For $G \in \mathbb{C}^3$, the plane wave $r \mapsto e^{i\langle G,r\rangle}$ is the solution of the equation $\frac{1}{2}\nabla_r^2\phi = \epsilon\phi$ with $\epsilon = \frac{1}{2}\|G\|^2$. The operator $\frac{1}{2}\nabla_r^2$ is the kinetic operator for an electron in atomic units. Thus the plane wave defined by G corresponds to the kinetic energy $\epsilon = \frac{\|G\|^2}{2}$ Hartree.

Therefore, in Abinit, the wave functions are discretized by choosing a few vectors $k \in \mathbb{R}^3$

called k-points and fixing a cut-off energy E_{cut} to expand the the Kohn-Sham's orbitals as

$$\phi(r,\sigma) = \sum_k \sum_G c_{k,G} e^{i\langle k+2\pi G, r\rangle} \tag{2.14}$$

with G in

$$\left\{ G \in \mathbb{R}^3 \text{ s.t. } G = n_1 b_1 + n_2 b_2 + n_3 b_3 \text{ with } n_1, n_2, n_3 \in \mathbb{Z} \text{ and } \|G\|^2 \leqslant \frac{2E_{\text{cut}}}{4\pi^2} \right\}.$$

Since each k-point should correspond to a different set of orbitals, they can each be treated separately.

Plane waves have many benefits: they are very well suited to the study of crystals, their completeness is systematic (the more plane waves you have, the more complete the basis is), the expression of the potentials projected onto plane waves are simply obtained by a Fourier transform (which is an actual benefit for the Hartree potential). But the main drawback is that the discretized Hamiltonian is a dense matrix, which would be very expensive to store and compute. In Abinit the matrix is never entirely computed. Only its application to a vector is computed when needed. Therefore, any algorithm used with the discretize H needs to be matrix-free.

2.5. Pseudopotential

Most plane-wave DFT calculations are made using pseudopotentials. This method is based on the assumption that only valence electrons contribute significantly to the physical and chemical properties of a given system. Therefore, core electrons are frozen in their atomic state, calculated once with DFT calculation on a isolated atom, and expressed in a basis of N_p projectors. Only valence electrons are explicitly treated in the DFT computation, which significantly reduces the complexity of the system to solve. We consider that the valence electrons evolve in a modified potential that accounts for interactions with the nucleus and core electrons. Since the core electron potential is difficult to write down in plane-wave form, we replace it with a pseudopotential that has the same effect on the valence electrons. This greatly reduces the number of projectors needed to represent the electrons.

Hence a new pseudopotential term is introduced in expression of the Hamiltonian operator with a local and non local part. Hence we can rewrite

$$H = T_s + V_{\text{loc}} + V_{\text{nonloc}} \tag{2.15}$$

where the external potential, Hartree potential and exchange and correlation potential, as

well as the local part of the pseudopotential, have been included in V_{loc}.

There are several techniques to write pseudopotentials, one of which is called PAW for Projector Augmented-Wave. This method is featured in Abinit and changes the eigenproblem $HU = U\Lambda$ to a generalized eigenproblem $HU = \Lambda BU$, where B is called the overlap operator. However, in most of this work, we will focus on standard eigenproblems.

2.6. Solving the Kohn-Sham equations with self-consistent field iterations

The system of discretized Kohn-Sham equations to solve is

$$H(U)U = U\Lambda \tag{2.16}$$

for $U \in \mathbb{C}^{N\times m}$ matrix with orthonormal columns, $\Lambda \in \mathbb{C}^{m\times m}$ a diagonal matrix with the m smallest eigenvalues of $H(U)$ counted with multiplicity and where $H : \mathbb{C}^{N\times m} \to C^{N\times N}$ associate to U the corresponding Hamiltonian matrix. Thus, it is a fix-point problem that can be solve using self-consistent iterations (see algorithm 1). We start with a random Hamiltonian matrix H_0, then compute its smallest eigenpairs. Using the eigenvectors, we can express the corresponding density ρ and the corresponding Hamiltonian operator $H_1 = H[\rho]$. We iterate the same process until convergence.

Algorithm 1 Self-consistent field in DFT

```
H ← RandomHamiltonian(N)
while H not converged do
    Λ, U ← EigenSolver(H)
    H ← Hamiltonian(U)
end while
```

The convergence of this algorithm is not guaranteed, actually it will most likely not converged in this naive version. It requires some mixing and preconditioning, but this will not be explored here as it is not the scope of this work.

Instead, we will focus on solving the eigenvalue problem efficiently. In Abinit, the diagonalization step is very expensive, especially since computing the application of the Hamiltonian to a vector is highly resource intensive. Moreover, we only need a limited set of eigenvalues (the smallest ones).

Therefore, to save computation time, we use iterative eigensolvers, to solve the problem inexactly. This type of solvers also have the advantage of being matrix-free, which is necessary. More precisely, we execute a few steps of the iterative eigensolver at each SCF

step starting from the current estimation. This way, the resolution of the eigenproblem is done more and more accurately as we approach convergence. This way, we avoid solving with high precision a problem with an inaccurate Hamiltonian. This modified SCF routine is presented in algorithm 2.

Algorithm 2 Double self-consistent field in DFT

$X \leftarrow$ RandomVectors(N, m)
$H \leftarrow$ Hamiltonian(U)
while H not converged **do**
  $\Lambda, X \leftarrow$ EigenSolverStep(H, X)
  $H \leftarrow$ Hamiltonian(U)
end while

3. Iterative Eigensolvers

In this section we will present two families of iterative algorithms to solve eigenvector problems: vector-based algorithms, which make use of the fact that the solution to the problem is also the solution to a constrained minimisation problem, and subspace-based algorithms, in which the search is not directly for vectors but for the eigensubspace.

First of all, let's define our problem. Let H be a hermitian matrix in $\mathbb{C}^{N\times N}$. We denote $\lambda_1, \dots, \lambda_N$, its eigenvalues, in increasing order and $u_1, \dots, u_N$ some corresponding eigenvectors. Since H is hermitian, $u_1, \dots, u_N$ can be chosen orthonormal. The objective is to find the m smallest eigenpairs $\{(\lambda_1, u_1), \dots, (\lambda_m, u_m)\}$. We call $\mathcal{U} = \text{Span}\{u_1, \dots, u_m\}$ the the eigensubspace of sought eigenvectors.

Both types of algorithms use the Rayleigh-Ritz procedure, that will be presented next.

3.1. The Rayleigh-Ritz method

The Rayleigh-Ritz method is a numerical method to approximate eigenvalues and eigenvectors of a matrix or operator. Given a matrix $H \in \mathbb{C}^{N\times N}$ and a matrix $X \in \mathbb{C}^{N\times m}$ with orthonormal columns, the Rayleigh-Ritz procedure is the following (also described in algorithm 3):

1. Compute $A = X^H H X$.
2. Solve the eigenproblem $Ay_i = \mu_i y_i$.
3. Compute the Ritz vectors $z_i = X y_i$.
4. The Ritz pair (μ_i, z_i) is an approximate solution to the eigenproblem of H.

This method gives approximate eigenvectors in the subspace described by the columns of X. If the subspace is already an eigensubspace, the method is exact.

Lemma 3.1.1. *If $\mathcal{X} = \text{Span}\{x_1, \dots, x_m\}$ is an eigensubspace of H, the Rayleigh-Ritz method on H with $X = (x_1|\dots|x_m)$ gives the exact eigenpairs of H contained in $\mathcal{X}$.*

Proof. Let y_i, μ_i such that $Ay_i = \mu_i y_i$ with $A = X^H H X$. Then with $z_i = X y_i$, we have

$$X X^H H z_i = \mu_i z_i.$$

The product XX^H is the orthogonal projector on $\mathcal{X} = \text{Span}\{x_1, \ldots, x_m\}$ which is stable by H since it is an eigensubspace. Thus, as $z_i \in \mathcal{X}$ and $Hz_i \in \mathcal{X}$ we have,

$$Hz_i = XX^H Hz_i = \mu_i z_i.$$

Conversely, if $Hz_i = \mu_i z_i$ with $z_i \in \mathcal{X}$, their is a $y_i \in \mathbb{C}^m$ such that $z_i = Xy_i$ and then, $X^H HXy_i = \mu_i X^H Xy_i = \mu_i y_i$ because $X^H X = I_m$ (X has orthonormal columns). □

Algorithm 3 Rayleigh-Ritz

function RAYLEIGHRITZ(H, X)
  $X \leftarrow$ ORTHO(X)
  $A \leftarrow X^H HX$
  $\lambda, Y \leftarrow$ EIGEN(A)
  $U \leftarrow XY$
  return λ, U
end function

3.1.1. Convergence of the Rayleigh-Ritz method

We will now study the convergence of Ritz pairs obtained from the Rayleigh-Ritz method, in the case where the subspace $\mathcal{X}$ spanned by the columns of X converges towards $\mathcal{U}$, following the work from [JS01]. To measure the distance between the two subspaces we will use the $\sin\angle(\cdot,\cdot)$ function defined bellow and we will denote $\varepsilon = \sin\angle(\mathcal{X},\mathcal{U})$.

Definition 3.1.1. *Let $\mathcal{V}$ and $\mathcal{W}$ two subspaces of $\mathbb{C}^N$, V the column matrix of an orthonormal basis of $\mathcal{V}$ and $W_\perp$ the column matrix of an orthonormal basis of $\mathcal{W}^\perp$. We define*

$$\sin\angle(\mathcal{V},\mathcal{W}) = \|{W_\perp}^H V\|_2. \tag{3.1}$$

Here and in the rest of this work, the norm $\|\cdot\|_2$ denote either the euclidean norm when applied to a vector or the operator norm associated to the euclidean norm when applied to a matrix. In this section, we denote Λ, the diagonal matrix with $\lambda_1, \ldots, \lambda_m$ on its diagonal. We also define $W = X^H U$ and $W_\perp = {X_\perp}^H U$, such that $U = XW + X_\perp W_\perp$. Then to orthonormalize the columns of W, we introduce

$$Q = \begin{pmatrix} W^H & W \end{pmatrix}^{-\frac{1}{2}}$$

such that, with $\hat{W} = WQ$ we have $\hat{W}^H\hat{W} = \hat{W}\hat{W}^H = I_m$.

To study convergence, we will need to use Elsner's theorem [Els85], that is recalled hereunder.

Theorem 3.1.1 (Elsner)**.** *For A and B in $\mathbb{C}^{n\times n}$, with respective spectra $\{\lambda_1,\dots,\lambda_n\}$ and $\{\mu_1,\dots,\mu_n\}$, we have*

$$\max_{1\leqslant j\leqslant n}\min_{1\leqslant i\leqslant n}|\lambda_i-\mu_j|\leqslant(\|A\|_2+\|B\|_2)^{1-\frac{1}{n}}\|A-B\|_2^{\frac{1}{n}}. \tag{3.2}$$

Convergence of Ritz Values

To begin with, we will study the convergence of Ritz values, using the following theorem from [JS01].

Theorem 3.1.2. *Let $A=X^HHX$ the matrix diagonalized in the Rayleigh-Ritz procedure. There is a matrix E in $\mathbb{C}^{m\times m}$ satisfying*

$$\|E\|_2\leqslant\frac{\varepsilon}{\sqrt{1-\varepsilon^2}}\|H\|_2 \tag{3.3}$$

such that $(Q^{-1}\Lambda Q,\hat{W})$ is an eigenblock of $A+E$.

Proof. The diagonal matrix Λ is such that,

$$HU-U\Lambda=0.$$

Then, with $X_\perp$ the matrix in $\mathbb{C}^{N\times N}$ whose columns form an orthonormal basis of $\mathcal{X}^\perp$, we get

$$X^HH\begin{pmatrix}X & X_\perp\end{pmatrix}\begin{pmatrix}X^H\\ {X_\perp}^H\end{pmatrix}U-X^HU\Lambda=0$$

which gives

$$AW+X^HHX_\perp W_\perp-W\Lambda=0.$$

Then, multiplying by Q from the right we get

$$A\hat{W}+X^HHX_\perp W_\perp Q-\hat{W}Q^{-1}\Lambda Q=0$$

and we define

$$S=\hat{W}Q^{-1}\Lambda Q-A\hat{W}$$

such that

$$\|S\|_2=\|X^HHX_\perp W_\perp Q\|_2\leqslant\|X^HHX_\perp\|_2\|W_\perp Q\|_2.$$

From the orthonormality of X and $X_\perp$, we have $\|X^H H X_\perp\|_2 \leqslant \|H\|_2$. Furthermore, $\varepsilon = \|{X_\perp}^H U\|_2 = \|W_\perp\|_2$ and $W^H W = I_m - {W_\perp}^H W_\perp$ so $\|W^H W| \geqslant 1 - \|W_\perp\|_2^2 = 1 - \varepsilon^2$. Thus

$$\|S\|_2 \leqslant \frac{\varepsilon}{\sqrt{1-\varepsilon^2}} \|H\|_2.$$

Now we define $E = S\hat{W}^H$, it is clear that $\|E\|_2 \leqslant \varepsilon \left(1-\varepsilon^2\right)^{-1/2} \|H\|_2$ and also

$$(A+E)\hat{W} = A\hat{W} - A\hat{W}\hat{W}^H\hat{W} + \hat{W}Q^{-1}\Lambda Q\hat{W}^H\hat{W} = \hat{W}Q^{-1}\Lambda Q$$

using $\hat{W}^H\hat{W} = I_m$. □

This allows us to deduce the following corollary.

Corollary 3.1.1. *Let $\mu_1, \ldots, \mu_m$ be the eigenvalues of $A = X^H H X$. Then there are integers $j_1 \ldots, j_m$ such that*

$$|\lambda_i - \mu_{j_i}| \leqslant 4 \left(\left(2 + \frac{\varepsilon}{\sqrt{1-\varepsilon^2}} \right) \|H\|_2 \right)^{1-\frac{1}{m}} \left(\frac{\varepsilon}{\sqrt{1-\varepsilon^2}} \|H\|_2 \right)^{\frac{1}{m}}. \tag{3.4}$$

Proof. The eingenvalues of $(A+E)$ are then the eigenvalues of Λ and using Elsner's theorem (3.1.1), we know that there is a permutation $j_1, \ldots, j_m$ of $1, \ldots, m$ such that

$$|\lambda_i - \mu_{j_i}| \leqslant 4(\|A\|_2 + \|A+E\|_2)^{1-\frac{1}{m}} \|E\|_2^{\frac{1}{m}}.$$

The conclusion comes from $\|A\|_2 < \|H\|_2$ and $\|E\|_2 \leqslant \varepsilon \left(1-\varepsilon^2\right)^{-1/2} \|H\|_2$. □

Now if we have a sequence $\{\mathcal{X}_n\}_{n \geqslant 0}$ (associated to a sequence $\{X_n\}_{n \geqslant 0}$ of matrices with orthogonal columns) of subspaces such that $\sin \angle(\mathcal{X}_n, \mathcal{U}) = \varepsilon_n \underset{n\infty}{\longrightarrow} 0$, we can write $\mu_1^n, \ldots, \mu_m^n$ the eigenvalues of $A_n = {X_n}^H H X_n$ such that for all $n \geqslant 0$ and $1 \leqslant i \leqslant m$,

$$|\lambda_i - \mu_i^n| \leqslant 4 \left(\left(2 + \frac{\varepsilon_n}{\sqrt{1-{\varepsilon_n}^2}} \right) \|H\| \right)^{1-\frac{1}{m}} \left(\frac{\varepsilon_n}{\sqrt{1-{\varepsilon_n}^2}} \|H\| \right)^{\frac{1}{m}}, \tag{3.5}$$

which implies $\mu_i^n \underset{n\infty}{\longrightarrow} \lambda_i$.

Convergence of Ritz pairs

Now we want to investigate the convergence of Ritz pairs, that we denote (μ_i^n, z_i^n) for $i = 1, \ldots, m$, returned by the Rayleigh-Ritz procedure. From the previous section we already now that $\mu_i^n \underset{n\infty}{\longrightarrow} \lambda_i$. Obviously, we can't expect z_i^n to converge towards u_i, because eigenvectors are not unique. To get around this lack of uniqueness, we will observe

the distances between eigensubspaces. We denote $\mathcal{U}_1, \dots \mathcal{U}_l$ the eigensubspaces associated to each *unique* eigenvalue of Λ and $\hat{\lambda}_1, \dots, \hat{\lambda}_l$ the associated eigenvalues such that for $j = 1, \dots, l$, the dimension of $\mathcal{U}_j$ is the multiplicity of $\hat{\lambda}_j$. Then for $j = 1, \dots, l$ we denote $\mathcal{Z}_j^n = \text{Span}\{z_i \text{ s.t. } \lambda_i = \hat{\lambda}_j\}$. We want to show that the distance (measured with $\sin\angle(\cdot,\cdot)$) between $\mathcal{U}_j$ and $\mathcal{Z}_j$ goes to zero.

To study this convergence, we will use the following modified version of Davis and Kahan $\sin\theta$ theorem.

Theorem 3.1.3. *Let A and B be two hermitian $m \times m$ matrices. We denote $\lambda_1 \geqslant \dots \geqslant \lambda_m$ and $\mu_1, \dots, \mu_m$ their respective eigenvalues and $u_1, \dots, u_m$ and $v_1, \dots, v_m$ some corresponding eigenvectors. We assume $\lambda_1 = \dots = \lambda_k$ and $\lambda_k > \lambda_{k+1}$. Then we denote $\mathcal{U}_1 = \text{Span}\{u_1, \dots, u_k\}$, $U_1 = (u_1|\dots|u_k)$, $\mathcal{V}_1 = \text{Span}\{v_1, \dots, v_k\}$ and $V_1 = (v_1|\dots|v_k)$ the eigensubspaces and matrices corresponding to the subspace associated with the eigenvalue λ_1 and $\mu_1, \dots, \mu_k$. Then we have*

$$\sin\angle(\mathcal{V}_1, \mathcal{U}_1)^2 \leqslant \sum_{i=1}^{k} \frac{|\lambda_1 - \mu_i| + \|B - A\|_2}{\lambda_1 - \lambda_{k+1}}. \tag{3.6}$$

Proof. Let x in $\mathbb{C}^m$ with norm 1. We denote $x = \sum_{i=1}^m \alpha_i u_i$ its expression in the basis $\{u_1, \dots, u_m\}$. Since x has norm 1, $\sum_{i=1}^m |\alpha_i|^2 = 1$. Then

$$\begin{aligned}
x^H A x &= \sum_{i=1}^{m} \lambda_i \alpha_i x^H u_i \\
&= \sum_{i=1}^{m} \lambda_i |\alpha_i|^2 \\
&= \lambda_1(|\alpha_1|^2 + \dots + |\alpha_k|^2) + \sum_{i=k+1}^{m} \lambda_i |\alpha_i|^2 \\
&\leqslant \lambda_1(|\alpha_1|^2 + |\dots| + |\alpha_k|^2) + \lambda_{k+1} \sum_{i=k+1}^{m} |\alpha_i|^2 \\
&= \lambda_1 - (\lambda_1 - \lambda_{k+1})(\sum_{i=k+1}^{m} |\alpha_i|^2).
\end{aligned}$$

Thus

$$\sum_{i=k+1}^{m} |\alpha_i|^2 \leqslant \frac{\lambda_1 - x^H A x}{\lambda_1 - \lambda_{k+1}}.$$

Now

$$\begin{aligned}
\sin\angle(\mathcal{V}_1,\mathcal{U}_1)^2 &= \|{U_2}^H V_1\|^2 \\
&= \sup_{\substack{v\in\mathcal{V}_1 \\ \|v\|_2=1}} \|{U_2}^H v\|^2 \\
&= \sup_{\substack{v\in\mathcal{V}_1 \\ \|v\|_2=1}} \sum_{j=k+1}^{m} |u_j^H v|^2 \\
&= \sup_{\substack{\beta_1,\dots\beta_k \\ |\beta_1|^2+\dots+|\beta_k|^2}} \sum_{i=1}^{k}\sum_{j=k+1}^{m} \beta_i |u_j^H v_i|^2 \\
&\leqslant \sum_{i=1}^{k}\sum_{j=k+1}^{m} |u_j^H v_i|^2 \\
&\leqslant \sum_{i=1}^{k} \frac{\lambda_1 - {v_i}^H A v_i}{\lambda_1 - \lambda_{k+1}} \\
&= \sum_{i=1}^{k} \frac{\lambda_1 - \mu_i - {v_i}^H (B-A) v_i}{\lambda_1 - \lambda_{k+1}} \\
&\leqslant \sum_{i=1}^{k} \frac{|\lambda_1 - \mu_i| + |{v_i}^H (B-A) v_i|}{\lambda_1 - \lambda_{k+1}}
\end{aligned}$$

Then, for $i = 1,\dots,k$, we have from Cauchy-Schwarz that $|{v_i}^H(B-A)v_i| \leqslant \|v_i\|_2\|(B-A)v_i\|_2 \leqslant \|B-A\|_2$. This concludes the proof. □

Now let j be in $\{1,\dots,m\}$, without loss of generality, we can assume that $\hat{\lambda}_j$ is the largest eigenvalue of $A^n + E^n$ ($A^n + E^n$ can be replaced with $(A^n + E^n - (\hat{\lambda}_j + \epsilon)I_m)^{-1}$ that has the same eigenvectors). We denote $z_{i_1}^n,\dots,z_{i_r}^n$ the eigenvectors in $\mathcal{Z}_j^n$ and $\mu_{i_1}^n,\dots,\mu_{i_r}^n$ the corresponding eigenvalues, such that

$$\sin\angle(\mathcal{Z}_j,\mathcal{U}_j)^2 \leqslant \sum_{k=1}^{r} \frac{|\hat{\lambda}_j - \mu_{i_k}^n| + \|E^n\|_2}{\delta} \tag{3.7}$$

where $\delta > 0$ is the difference between $\hat{\lambda}_j$ and its closest distinct eigenvalue. Since $|\hat{\lambda}_j - \mu_{i_k}^n| \xrightarrow{n\infty} 0$ and $\|E^n\|_2 \xrightarrow{n\infty} 0$, we have

$$\sin\angle(\mathcal{Z}_j,\mathcal{U}_j) \xrightarrow{n\infty} 0. \tag{3.8}$$

This means that the subspaces returned by the Rayleigh-Ritz procedure indeed converges

towards the subspaces associated to each distinct eigenvalue of H in $\mathcal{U}$ provided that $\mathcal{X}$ tends towards $\mathcal{U}$.

3.2. Minimization algorithms

Let $H(U)$ be the discretized Hamiltonian operator corresponding to $U \in \mathbb{C}^{N\times m}$. The discretized Kohn-Sham's equation can be written as the non-linear eigenvalue problem

$$\begin{cases} H(U)U = U\Lambda \\ U^H U = I_m \end{cases} \tag{3.9}$$

for $U \in \mathbb{C}^{N\times m}$ and $\Lambda \in \mathbb{C}^{m\times m}$ a diagonal matrix with the m smallest eigenvalues of $H(U)$.

Now we consider the constrained minimization problem

$$\begin{aligned} &\min_{U\in\mathbb{C}^{N\times m}} E(U) \\ &\text{s.t. } U^H U = I_m \end{aligned} \tag{3.10}$$

where $E(U)$ is the energy defined as $E(U) = \mathrm{tr}(U^H H(U)U)$. The Lagrangian function associated with this optimization problem is

$$\mathcal{L}(U,\Lambda) = E(U) - \mathrm{tr}(\Lambda(U^H U - I_m))$$

and its gradient with respect to U and Λ is

$$\nabla\mathcal{L}(U,\lambda) = \begin{pmatrix} H(U)U - U\Lambda \\ U^H U - I_m \end{pmatrix}.$$

We see that if (U,Λ) is a solution of (3.9), it satisfies

$$\nabla\mathcal{L}(U,\Lambda) = 0. \tag{3.11}$$

Thus solutions of the minimization problem are solutions of the eigenvalue problem. Conversely if V is a global minimizer of (3.12), there is a unitary matrix $R \in \mathbb{C}^{m\times m}$ such that $V = UR$ where U solves (3.9). Indeed, the matrix $V^H HV \in \mathbb{C}^{m\times m}$ is Hermitian so there is a unitary matrix $R \in \mathbb{C}^{m\times m}$ so that $R^H V^H HVR = M$ with $M \in \mathbb{C}^{m\times m}$ diagonal. The eigenvalues $\mu_1, \ldots, \mu_m$ of M are eigenvalues of H and since $E(V) = \mu_1 + \ldots + \mu_m$, they must be the m smallest one (otherwise the solutions of (3.9) give smaller energies). Thus VR is a solution of the non linear eigenvalue problem. This means that the subspaces

associated to the solutions of both problems are the same.

Thus, iterative minimization algorithms can be used to solve Kohn-Sham equations in their minimization form. Since the objective function E is quite complicated, minimization algorithms are not used directly on E. Instead, we use self-consistent iterations, that is at each iteration, the Hamiltonian operator $H(X_k)$ is fixed so that the minimization problem becomes quadratic :

$$\begin{aligned} \min_{U \in \mathbb{C}^{N \times m}} & \operatorname{tr}(U^H H(X_k) U) \\ \text{s.t. } & U^H U = I_m \end{aligned} . \tag{3.12}$$

Then a few iterations of the chosen minimization technique are applied, starting from X_k and yield the new estimate X_{k+1}, which is then used again at the next iteration to fix the Hamiltonian. This is repeated until convergence.

We will present here two iterative minimization algorithms that are commonly used in DFT calculations: conjugate gradient and LOBPCG algorithm.

3.2.1. Conjugate gradient

The conjugate gradient algorithm solves quadratic programming problems

$$\min_{x \in \mathbb{C}^N, \, \|x\|=1} x^H A x - x^H b \tag{3.13}$$

where $A \in \mathbb{C}^{N \times N}$ is hermitian positive definite and b is a vector $\mathbb{R}^N$. The idea is to find a basis $\{p_1, \ldots, p_N\}$ of $\mathbb{C}^N$ whose elements are mutually conjugated with respect to A i.e. $p_i^H A p_j = \delta_{ij}$ for all $1 \leqslant i, j \leqslant N$. Then the optimal solution x^* can be written $x^* = \sum_{i=1}^N \alpha_i p_i$. We note that the optimal solution x^* is such that the gradient of $f : x \mapsto x^H A x - x^H b$ is zero i.e. $Ax^* - b = 0$. This gives $\alpha_i = \frac{{p_i}^H b}{{p_i}^H A p_i}$. If the directions p_i are well chosen, we don't need to have a complete basis of $\mathbb{C}^N$ to have a good approximation of x^*. The *iterative* conjugate gradient algorithm consists in building the directions p_i iteratively in order to reduce f as much as possible at each step. At step $i \geqslant 0$, we have an estimate $x_i = \sum_{j=0}^i \alpha_j p_j$ (wlog. $x_0 = 0$) and we compute $r_i = b - A x_i$ which is the opposite of the gradient of f at x_i and thus indicate the direction of steepest descent. Then, since we want the new direction p_{i+1} to to be conjugate to the already computed directions, we take

$$p_{i+1} = r_i - \sum_{j \leqslant i} \frac{{r_i}^H A p_j}{{p_j}^H A p_j} p_j .$$

This gives some simple recurrence formula, that allow us to compute a new iteration from the previous one only. Another benefit of this method is that it allows preconditioning.

The use of the iterative conjugate gradient algorithm in DFT computation is described in [PTA+92] and [KF96]. In the case of DFT calculations, we want to solve

$$\min_{X \in \mathbb{C}^{N \times m}} X^H H X \tag{3.14}$$

with an orthonormality condition

$$X^H X = I_m. \tag{3.15}$$

To solve this, we apply m successive conjugate gradient algorithms, with the additional condition that an estimated x^n $(1 \leqslant n \leqslant m)$ solution to (3.14) must also be orthogonal to the already computed estimate solutions $x^1, \ldots, x^{n-1}$. This is done by forcing the directions p_i in the conjugate gradient algorithm to also be orthogonal to $x^1, \ldots, x^{n-1}$ with

$$p_{i+1} = r_i - \sum_{j=0}^{i} \frac{{r_i}^H H p_j}{{p_j}^H H p_j} p_j - \sum_{k=1}^{n-1} \frac{r_i^H x^k}{{x^k}^H x^k} x^k.$$

This modified algorithm is called the projected conjugate gradient algorithm. In practice, at each SCF step, a fixed number of directions is chosen for each chosen eigenpairs, starting with the result of the previous iteration.

This method does not directly gives the solution to the eigenproblem but a basis of the solution subspace $\mathcal{U}$. To retrieve the solution, we need to apply the Rayleigh-Ritz method on $X = (x^1 | \ldots | x^m)$ once each vector has been updated.

3.2.2. LOBPCG

The Locally-Optimal Block Preconditioned Conjugate Gradient (LOBPCG) algorithm is an evolution of the Conjugate Gradient algorithm developed by Knyazev in [Kny01]. This method is implemented in Abinit as the default eigensolver and is the most commonly used.

The term "Block" in LOBPCG means that several vectors are computed simultaneously in blocks. That is we replace the vectors x_i and p_i from above with some matrices X_i and P_i containing several vectors. Each vector of X_i should approximate one of the sought eigenvectors. The matrix of directions P_i is computed similarly to the conjugate gradient case and used to update X_i. Then we need to apply the Rayleigh-Ritz method to compute the next estimation X_{i+1}. The computation in blocks allows calculations to be parallelized.

The "Locally-Optimal" part of the name refers to the fact that the two dimensional optimization method used to determine α_{i+1} such that $x_{i+1} = x_i + \alpha_{i+1} p_{i+1}$ minimizes the residual is replaced with a 3 dimensional optimization problem on X_i, X_{i-1} and P_{i+1}. This speeds up the convergence.

Lastly, the LOBPCG, like the conjugate gradient method can be very efficiently precon-

ditioned, which also speeds up the convergence.

Once all the blocks $X^1, \ldots, X^{m'}$ are updated, we need to apply the Rayleigh-Ritz method on $(X^1_{i+1}|\ldots|X^{m'}_{i+1})$ to retrieve the actual eigenvectors of H, just like in the conjugate gradient method. This method works very well and converged fast due to preconditionning. Its major drawback is the numerous calls to the Rayleigh-Ritz procedure.

3.3. Subspace iteration algorithms

In subspace iteration methods, we want to estimate the subspace $\mathcal{U}$, rather than the vectors $u_1, \ldots, u_m$ directly. To do so, we begin with a random initial subspace $\mathcal{X}_0 = \{x^0_1, \ldots, x^0_m\}$ and the for $k \geqslant 0$, we compute

$$\mathcal{X}_{k+1} = f(\mathcal{X}_k) = \text{Span}\{f(x^k_1), \ldots, f(x^k_m))\} \tag{3.16}$$

where $f : \mathbb{C}^N \to \mathbb{C}^N$ is called the filter and should be chosen to enhance the component in $\mathcal{U}$ and reduce the component in $\mathcal{U}^\perp$ (or any other complementary subspace) of every vectors in $\mathbb{C}^N$. Then, for an estimate $\mathcal{X}_k$ of $\mathcal{U}$, we can retrieve an estimate of the eigenvectors $u_1, \ldots, u_m$ via the Rayleigh-Ritz method.

If the subspace iteration method is applied to a constant matrix, the Rayleigh-Ritz step can be done only once when $\mathcal{X}_k$ has converged, as illustrated in algorithm 4. However, due to numerical instabilities it might be best to use the Rayleigh-Ritz step more often to ensure that the basis vectors of $\mathcal{X}_k$ remain linearly independent. Even thought mathematically the Rayleigh-Ritz step does not change the subspace, it makes a big difference numerically. Now, when the method is used in the self-consistent field algorithm, we need an estimate of $u_1, \ldots, u_m$ at each SCF step to update H so the Rayleigh-Ritz step must be done at each iteration as illustrated in algorithm 5.

Algorithm 4 Subspace iteration algorithm on a constant matrix H

```
X ← RandomVectors(N, m)
while X not converged do
    X ← f(X)
end while
λ, X ← RayleighRitz(H, X)
```

Algorithm 5 Subspace iteration algorithm in the self-consistent field

```
X ← RandomVectors(N, m)
H ← Hamiltonian(X)
while H not converged do
    X ← f(X)
    λ, X ← RayleighRitz(H, X)
    H ← Hamiltonian(X)
end while
```

3.3.1. Filters

The filter f should be a function from $\mathbb{C}^N$ to $\mathbb{C}^N$ enhancing or maintaining the components in $\mathcal{U}$ of each vector and reducing their components in $\mathcal{U}^\perp$. Also, the application of f to a vector in $\mathbb{C}^N$ should be easy and fast to compute.

Ideally, the range of f would be $\mathcal{U}$ and its kernel would be a complementary subspace of $\mathcal{U}$, so that $\mathcal{X}_1 = \mathcal{U}$. For instance, the orthogonal projector p onto U is ideal. However, such a filter can only be build with previous knowledge of $\mathcal{U}$. In the following, we will describe polynomial and rational filters whose main benefit is that they only require the knowledge of the eigenvalues $\lambda_1, \ldots, \lambda_N$ of H.

Let v be a vector in $\mathbb{C}^N$. It can be expressed in the orthonormal basis $\{u_1, \ldots, u_N\}$:

$$v = \sum_{i=1}^{N} \alpha_i u_i$$

with $\alpha_1, \ldots, \alpha_N \in \mathbb{C}$. Then since, for $n \geqslant 1$,

$$H^n u_i = H^{n-1} \lambda_i u_i = \ldots = {\lambda_i}^n u_i$$

we have for P a polynomial in $\mathbb{C}[X]$,

$$P(H) u_i = P(\lambda_i) u_i$$

and therefore,

$$P(H) v = \sum_{i=1}^{N} P(\lambda_i) \alpha_i u_i.$$

The same goes for rational function. For $n \geqslant 1$,

$$u_i = \frac{1}{\lambda_i} \lambda_i u_i = \frac{1}{\lambda_i} H u_i = \frac{1}{{\lambda_i}^n} H^n u_i$$

so

$$H^{-n}u_i = \frac{1}{{\lambda_i}^n}u_i.$$

Thus, for a rational function $Q \in \mathbb{C}(X)$, $Q(H)u_i = Q(\lambda_i)u_i$ and

$$Q(H)v = \sum_{i=1}^{N} Q(\lambda_i)\alpha_i u_i.$$

This way, if we choose a polynomial or rational function Q that is small on $[\lambda_{m+1}, \lambda_N]$ and large on $[\lambda_1, \lambda_m]$, the filter $f = Q(H)$ will reduce the components of v in $\mathcal{U}^\perp$ and enhance those in $\mathcal{U}$. More precisely, the component in u_i $(i = 1, \ldots, N)$ of any vector will be increased or reduced by a factor $Q(\lambda_i)$ when the filter is applied.

The most straightforward example of polynomial filter is the power method, that is used to compute the largest (in absolute value) eigenvalue of the matrix H, using the fact that the eigenvalues of H^n are the eigenvalues of H to the power n. When n is big, only the largest eigenvalue dominate. Thus the eigenvector is approximated with

$$x_n = H^n x_0$$

and the approximated eigenvalue is given by the Rayleigh quotient

$$\mu_n = \frac{{x_n}^H H x_n}{{x_n}^H x_n} = \frac{\sum_i {\lambda_i}^{n+1}|\alpha_i|^2}{\sum_i {\lambda_i}^n |\alpha_i|^2} \xrightarrow{n\infty} \underset{\lambda=\lambda_1,\ldots,\lambda_N}{\arg\max} |\lambda|$$

with $x_0 = \sum_{i=1}^{N} \alpha_i u_i$. This correspond to the subspace iteration method with simply H as the filter and a wanted subspace of dimension 1 (meaning the Rayleigh-Ritz step is not needed).

3.3.2. Convergence of subspace iteration algorithms

In this section, we will investigate the convergence properties of the subspace iteration method, following the work from [Saa11] and [Saa16].

To begin with, we state theorem 3.3.1, from [Saa16] that gives us the convergence rate of the subspace iteration method when applied to a constant matrix.

Theorem 3.3.1. *Let $A \in \mathbb{C}^{N\times N}$ be an hermitian matrix, we denote $\gamma_1, \ldots, \gamma_N$ its eigenvalues ordered so that that $|\gamma_1| \geqslant \ldots \geqslant |\gamma_N|$ and $v_1, \ldots, v_N$ some corresponding orthogonal eigenvectors. Then we denote $\Pi = \sum_{i=1}^{m} v_i {v_i}^H$ the projector onto* $\mathrm{Span}\{v_1, \ldots, v_m\}$ *orthog-*

onally to $\mathrm{Span}\{v_{m+1}, \dots, v_N\}$. *Let* $\mathcal{X}$ *be a subspace of* $\mathbb{C}^N$ *of dimension* $m \geqslant 1$, *such that* $\dim \Pi\mathcal{X} = m$, *we denote* $\tilde{\mathcal{X}} = A\mathcal{X}$ *the subspace* $\mathcal{X}$ *filtered by* A.

Then, for each vector v_i, *with* $1 \leqslant i \leqslant m$, *there is a unique vector* $x_i \in \mathcal{X}$ *such that* $\Pi x_i = v_i$. *In addition,* $\tilde{x}_i = \frac{1}{\gamma_i} A x_i \in \tilde{\mathcal{X}}$ *satisfies* $\Pi \tilde{x}_i = v_i$ *and*

$$\|\tilde{x}_i - v_i\|_2 \leqslant \frac{|\gamma_{m+1}|}{|\gamma_i|} \|x_i - v_i\|_2. \tag{3.17}$$

Proof. Let $1 \leqslant i \leqslant m$. $\Pi\mathcal{X}$ has dimension m, therefore, by dimension equality, $\Pi\mathcal{X} = \mathrm{Ran}(\Pi)$ so $v_i \in \Pi\mathcal{X}$ i.e. there is a x_i in $\mathcal{X}$ such that $v_i = \Pi x_i$. We denote $w_i = (I_N - \Pi)x_i$, such that $x_i = v_i + w_i$. Then, we define

$$\tilde{x}_i = \frac{1}{\gamma_i} A x_i = \frac{1}{\gamma_i} A(v_i + w_i) = v_i + \frac{1}{\gamma_i} A w_i.$$

We have $\Pi\tilde{x}_i = \Pi v_i + \frac{1}{\gamma_i} \Pi A w_i$ so from $\Pi A = A\Pi$, we get

$$\Pi \tilde{x}_i = v_i.$$

Now, $\|\tilde{x}_i - v_i\|_2 = \|\frac{1}{\gamma_i} A w_i\|_2$ and we have $(I_N - \Pi)w_i = w_i$ and $\Pi A = A\Pi$ so $A w_i = (I_N - \Pi)A(I_N - \Pi)w_i$. Thus,

$$\|\tilde{x}_i - v_i\|_2 \leqslant \frac{1}{|\gamma_i|} \|(I_N - \Pi)A(I_N - \Pi)\|_2 \|w_i\|_2 = \frac{|\gamma_{m+1}|}{|\gamma_i|} \|x_i - v_i\|_2$$

because the spectrum of $(I_N - \Pi)A(I_N - \Pi)$ is $\{0, \gamma_{m+1}, \dots, \gamma_N\}$. □

This theorem shows that the orthogonal distance between the subspace $\tilde{\mathcal{X}}$ and v_i is multiplied by a factor $\mu = \frac{|\gamma_{m+1}|}{|\gamma_i|}$ compared to the orthogonal distance between $\mathcal{X}$ and v_i. This factor μ is called the convergence rate and convergence is ensured as long as $\mu < 1$. In our case, the matrix A from the theorem is the filter $Q(H)$ for a rational (or polynomial) function Q so the convergence rate is given by the quotient

$$\mu = \max_{m+1 \leqslant j \leqslant N} \frac{|Q(\lambda_j)|}{|Q(\lambda_i)|}. \tag{3.18}$$

Thus we want to choose a filter that maximizes the quotient between the biggest unwanted filtered eigenvalue and the smallest wanted one.

In the case of self-consistent iterations, the matrix H is updated at each iteration with the current estimate of the eigenvectors. Therefore, the above convergence analysis cannot be directly applied, but needs to be modified to account for the change in H, which creates

a change in the filter, at each iterations. This analysis was done in [Saa16] from which the following theorem is also taken.

Theorem 3.3.2. *Let $A \in \mathbb{C}^{N\times N}$ an hermitian matrix and $E \in \mathbb{C}^{N\times N}$ an hermitian perturbation matrix, we denote $\gamma_1, \dots, \gamma_N$ the eigenvalues of A ordered so that $|\gamma_1| \geqslant \dots \geqslant |\gamma_N|$ and $v_1, \dots, v_N$ some corresponding orthogonal eigenvectors. We also denote $\Pi = \sum_{i=1}^{m} v_i v_i^H$ the projector on $\mathrm{Span}\{v_1, \dots, v_m\}$ orthogonally to $\mathrm{Span}\{v_{m+1}, \dots, v_N\}$. Let $\mathcal{X}$ be a subspace of $\mathbb{C}^N$ of dimension $m \geqslant 1$, such that $\dim \Pi\mathcal{X} = m$, we denote $\tilde{\mathcal{X}} = (A+E)\mathcal{X}$ the subspace $\mathcal{X}$ filtered by $(A+E)$ and K the projector on $\tilde{X}$ orthogonally to $\mathcal{U} = \mathrm{Ran}\, P$.*

Then, for each vector v_i, for $1 \leqslant i \leqslant m$, there is a unique vector $x_i \in \mathcal{X}$ such that $\Pi x_i = v_i$. In addition, there is a $\tilde{x}_i \in \tilde{\mathcal{X}}$ that satisfies $\Pi\tilde{x}_i = v_i$ and

$$\|\tilde{x}_i - v_i\|_2 \leqslant \frac{|\gamma_{m+1}| + \|(I-K)E\|_2}{|\gamma_i|}\|x_i - v_i\|_2 + \frac{\|(I-K)Ev_i\|_2}{|\gamma_i|}. \tag{3.19}$$

Proof. Let $1 \leqslant i \leqslant 0$. The proof of the existence of x_i is the same as in theorem 3.3.1. Now with $\hat{x}_i = \frac{1}{\gamma_i}(A+E)x_i$, we have

$$\Pi\hat{x}_i = \Pi\frac{1}{\gamma_i}Ax_i + \Pi\frac{1}{\gamma_i}Ex_i = v_i + \Pi\frac{1}{\gamma_i}Ex_i$$

so we will define a correction term $f = K\frac{1}{\gamma_i}Ex_i$ so that with $\tilde{x}_i = \hat{x}_i - f \in \tilde{\mathcal{X}}$,

$$\Pi\tilde{x}_i = v_i + \Pi(I_N - K)\frac{1}{\gamma_i}Ex_i = v_i$$

since $\mathrm{Ran}(I_N - K) = \ker\Pi$. Then

$$\begin{aligned}\|\tilde{x}_i - v_i\|_2 &= \|\frac{1}{\gamma_i}(A+E)(x_i - v_i) - K\frac{1}{\gamma_i}Ex_i\|_2 \\ &\leqslant \frac{|\gamma_{m+1}| + \|(I-K)E\|_2}{|\gamma_i|}\|x_i - v_i\|_2 + \frac{\|(I-K)Ev_i\|_2}{|\gamma_i|}.\end{aligned}$$

□

The bound given by this theorem on the distance between $\tilde{X}$ and v_i is degraded compared to theorem 3.3.1, but we will show that if the perturbation matrix E goes to 0, convergence remains ensured, provided that the projector $(I_N - K)$ is bounded.

From here, we denote with upper index n the value of E, K, $\mathcal{X}$ and x_i at iteration n (with $\mathcal{X}^{n+1} = (A + E^{n+1})\mathcal{X}^n$). Using $\|v_i\| = 1$ we have

$$\|x_i^n - v_i\|_2 \leqslant \frac{|\gamma_{m+1}| + \|(I-K^n)E^n\|_2}{|\gamma_i|}\|x_i^{n-1} - v_i\|_2 + \frac{\|(I-K^n)E^n\|_2}{|\gamma_i|},$$

which can be rewritten, for $n \geqslant 1$,

$$d_n \leqslant (\alpha + \beta_n) d_{n-1} + \beta_n$$

using the notations $d_n = \|x_i^n - v_i\|_2$, $\beta_n = \frac{\|(I-K^n)E^n\|_2}{\gamma_i}$ and $\alpha = \frac{|\gamma_{m+1}|}{|\gamma_i|}$. From the assumption that E^n goes to 0 and $(I_N - K^n)$ is bounded, we have $\beta_n \xrightarrow{n\infty} 0$ and since $\alpha < 1$, there is a $\gamma < 1$ and $n_0 > 1$ such that, for all $n \geqslant n_0$,

$$\alpha + \beta_k < \gamma.$$

Then, for $n \geqslant n_0$,

$$d_n \leqslant \beta_n + \beta_{n-1}\gamma + \ldots + \gamma^{n-n_0}\beta_{n_0} \leqslant \sum_{k=n_0}^{p} \beta_k \gamma^{n-k} + \sum_{k=p+1}^{n} \beta_k \gamma^{n-k} \tag{3.20}$$

for any $n_0 \leqslant p \leqslant n$. For $\varepsilon > 0$, there is a p_0 such that for all $n \geqslant p_0$, $\beta_n \leqslant \varepsilon(1-\gamma)$. This yield, for the second term in (3.20),

$$\sum_{k=p_0+1}^{n} \beta_k \gamma^{n-k} \leqslant \varepsilon(1-\gamma) \sum_{k=p+1}^{n} \gamma^{n-k} \leqslant \varepsilon.$$

Since the sequence $\{\beta_k\}_k$ goes to zero, it has an upper bound β that we use to bound the first term in (3.20).

$$\sum_{k=n_0}^{p_0} \beta_k \gamma^{n-k} \leqslant \beta \sum_{k=n_0}^{p_0} \gamma^{n-k} = \beta\gamma^{n-p_0} \frac{1-\gamma^{p_0-n_0+1}}{1-\gamma} \leqslant \beta \frac{\gamma^{n-p_0}}{1-\gamma} \xrightarrow{n\infty} 0.$$

Thus for big enough n, this term is smaller than ε. This proves that $\|x_i^n - v_i\|_2$ goes to zero.

This ensure that if the self-consistent iterations are convergent. Thus the iterative subspace method will also converge. And, for the speed of convergence, unless the Hamiltonian operator converges linearly with a convergence rate smaller than $\frac{|\gamma_{m+1}|}{|\gamma_i|}$, its speed of convergence will dominate the convergence of the eigenvalues. This means that improving the convergence of the iterative eigensolver above a certain threshold is not useful.

3.3.3. Chebyshev-filtered subspace iteration

Chebyshev-filtered subspace iteration [ZSTC06] is an example of a subspace iteration method using Chebyshev polynomials as filters. The method is implemented in Abinit and yield very good results, with a better scalability than LOBPCG, as studied in [LT15].

Chebyshev filtering is used to determine the smallest (or largest) eigenpairs of a Hermitian matrix. Chebyshev polynomials of the first kind are such that $\cos(k\theta) = T_k(\cos\theta)$ and $\cosh(k\theta) = T_k(\cosh\theta)$. Therefore, $|T_k(t)| \leqslant 1$ for $t \in [-1, 1]$ and $|T_k|$ grows rapidly outside of this interval. Some Chebyshev polynomials of various degree are displayed in figure 3.1. This makes it possible to construct filters using translated Chebyshev polynomials, where the unwanted part of the spectrum $[\lambda_{m+1}, \lambda_N]$ is sent to $[-1, 1]$ to be attenuated while the rest of the spectrum is amplified. Thus Chebyshev filters are defined as follows

$$P_k(X) = T_k\left(\frac{1}{r}(X - c)\right) \tag{3.21}$$

with $r = \frac{1}{2}(a - b)$ and $c = \frac{1}{2}(a + b)$ for some $a, b \in \mathbb{R}$ such that $b > \lambda_N$ and $a > \lambda_m$. Since Chebyshev polynomials are monotonic outside $[-1, 1]$, we don't necessarily need to have $\lambda_m < a < \lambda_{m+1}$.

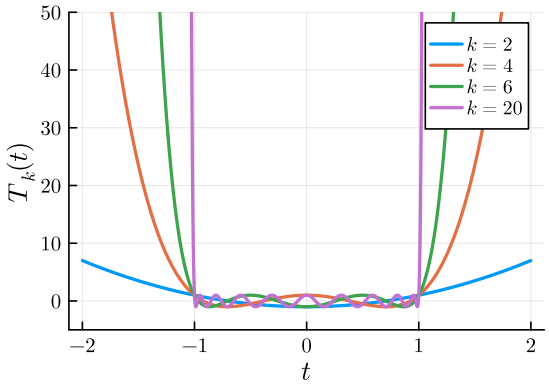


Figure 3.1.: Chebyshev polynomials T_k of degree $k = 2$, 4 6 and 10.

To implement the method, we need to determine the interval $[a, b]$ to be translated on $[-1, 1]$, where b is an upper bound of the largest eigenvalue λ_N and a is an upper bound of the largest sought eigenvalue λ_m. In the general case, the bound b can be determined using a few iterations of the power method. However, in plane-wave based DFT, an upper bound is readily available. The spectrum of the discretized Hamiltonian is bounded by the input parameter E_{cut} which is the maximal energy of the plane waves on which the operator is discretized. For the bound a, we can take the largest Rayleigh quotient of the current estimate, which is the largest eigenvalue of the matrix H projected in the current subspace estimate (of dimension m).

Algorithms 6 and 7 describes Chebyshev-filtered subspace iteration for a constant matrix and in the self-consistent loop. The recursive definition of Chebyshev polynomials allow us

to compute the filters efficiently.

Algorithm 6 Chebyshev filtering algorithm on a constant matrix H

$X \leftarrow$ RandomVectors(N, m)
$\Lambda, X \leftarrow$ RayleighRitz(H, X)
$b \leftarrow$ UpperBound(H)
while X not converged **do**
  $a \leftarrow \max \Lambda$
  $c \leftarrow \frac{b+a}{2}$, $r \leftarrow \frac{b-a}{2}$
  $P_0(H)X \leftarrow X$
  $P_1(H)X \leftarrow \frac{1}{r}(H - cI)X$
  for $l = 2$ to k **do**
    $P_l(H)X = \frac{2}{r}(HP_{k-1}(H)X - cP_{k-1}(H)X) - P_{k-2}(H)X$
  end for
  $X \leftarrow P_k(H)X$
  $\Lambda, X \leftarrow$ RayleighRitz(H, X)
end while

Algorithm 7 Chebyshev filtering algorithm in the self-consistent field

$X \leftarrow$ RandomVectors(N, m)
$\Lambda, X \leftarrow$ RayleighRitz(H, X)
$H \leftarrow$ Hamiltonian(X)
$b \leftarrow E_{\text{cut}}$
while H not converged **do**
  $a \leftarrow \max \Lambda$
  $X \leftarrow$ChebyshevFilter(H, a, b, X)
  $\Lambda, X \leftarrow$ RayleighRitz(H, X)
  $H \leftarrow$ Hamiltonian(X)
end while

Chebyshev polynomials are not the only type of polynomials that are small on an interval and rapidly growing outside. For instance, monomials X^k could also be good candidates, with a simpler one-term recurrence formula. It turns out that Chebyshev polynomials have some optimal properties that make the very good for subspace methods. This is described in [Saa11] from which the following theorem is taken.

Theorem 3.3.3. *Let $[a, b]$ be a non-empty interval in $\mathbb{R}$ and let γ be any real scalar with $\gamma > b$. then the minimum*

$$\min_{P \in \mathbb{R}_k[X],\, P(\gamma)=1} \max_{t \in [a,b]} |P(t)| \tag{3.22}$$

where $\mathbb{R}_k[X]$ denote the set of real polynomials of degree smaller or equal k, is reached by

the polynomial

$$C_k(X) = \frac{T_k\left(1 + 2\frac{t-b}{b-a}\right)}{T_k\left(1 + 2\frac{\gamma-b}{b-a}\right)}. \tag{3.23}$$

The proof of this theorem can be found in [Che82]. In the subspace iteration method, with a filter $P(H)$ where $P \in \mathbb{R}[X]$, the convergence rate of the i-th eigenpair is given by

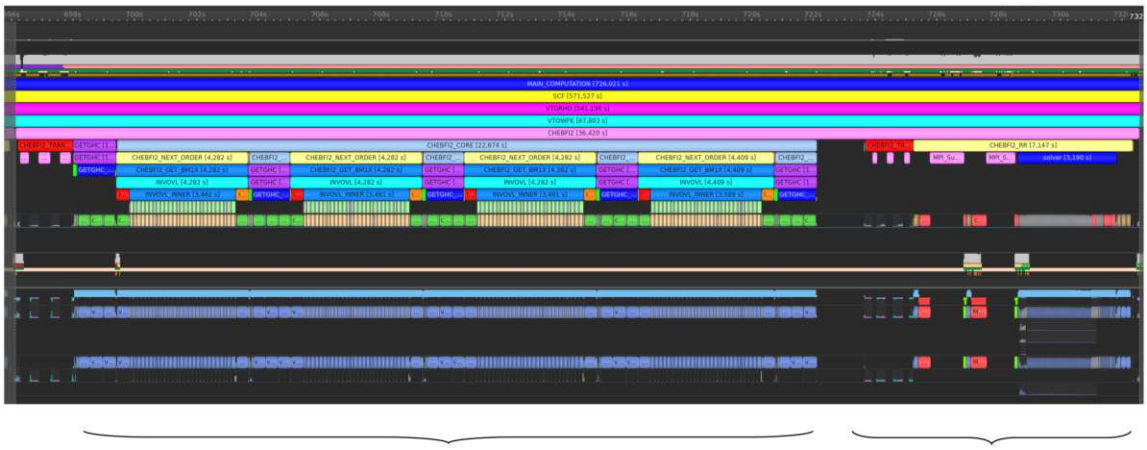
$$\max_{m+1 \leqslant j \leqslant N} \frac{|P(\lambda_j)|}{|P(\lambda_i)|}.$$

Without precise knowledge of the position of the eigenvalues, this can be approximated by

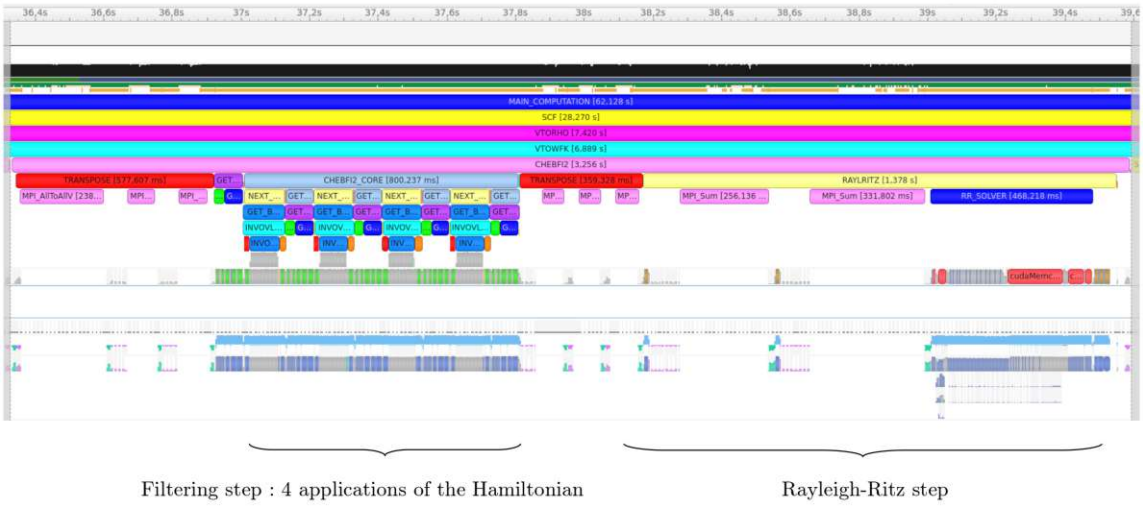
$$\max_{t \in [\lambda_{m+1}, \lambda_N]} \frac{P(t)}{P(\lambda_i)}$$

which is precisely minimal for the shifted Chebyshev polynomial, for a given maximal degree. This property, together with their definition by recurrence, which makes the calculation very practical, justifies the choice of Chebyshev polynomials.

This method has many benefits, the first one being that it works well with very low degree polynomials. For instance, the default degree in Abinit is 4. In addition, it has good scalability, in particular because the filter can be applied to the different vectors in parallel. This has enabled a significant improvement in performances when moving from CPU to GPU supercomputers, as illustrated in figure 3.2. Unfortunately, this evolution also highlighted a new bottleneck that is the Rayleigh-Ritz step, for which increasing the number of processors is of no help. A potential way to overcome this problem is the use of Slicing methods that will be presented in the next section.



(a) CPU



(b) GPU

Figure 3.2.: Time spent in the filtering step and in the Rayleigh-Ritz step during one SCF iteration with the Chebyshev filtering method, on CPU and GPU.

4. Slicing methods

This section is the main part of this work, in which we will present in detail the Spectrum Slicing method, whose objective is to get around the lack of parallelizability of the Rayleigh-Ritz step. The idea of Spectrum Slicing is to apply the Rayleigh-Ritz procedure to smaller subspaces, in parallel. This smaller subspaces should be eigensubspaces (or approximations of) so that the vectors obtained on each of them are directly eigenvectors of H. To obtain such subspaces, the spectrum of H is split into n_s slices $[\sigma_1, \sigma_2], [\sigma_2, \sigma_3], \ldots, [\sigma_{n_s}, \sigma_{n_s+1}]$ chosen so that

$$[\lambda_1, \lambda_m] \subset [\sigma_1, \sigma_2] \cup \ldots \cup [\sigma_{n_s}, \sigma_{n_s+1}].$$

Then, filtering methods are used to approximate the subspaces

$$\mathcal{U}_i = \text{Span}\{u_j \mid \lambda_j \in [\sigma_i, \sigma_{i+1}]\}$$

that are such that $\mathcal{U} = \mathcal{U}_1 \oplus \ldots \oplus \mathcal{U}_{n_s}$.

In short we want to apply some subspace iteration method to determine each subspace $\mathcal{U}_i$ with $1 \leqslant i \leqslant n_s$ and then reconstruct $\mathcal{U}$. The eigensubspace are chosen based on the spectrum since we know how to build polynomial or rational filters from it. Each $\mathcal{U}_i$ can be treated individually and the Rayleigh-Ritz step only needs to be applied in smaller dimension. Ideally, we would have to execute n_s Rayleigh-Ritz procedures in dimension $\frac{m}{n_s}$ in parallel. Since the time complexity of Rayleigh-Ritz is cubic, this can yield to some substantial gain. However we will see that this gain in the Rayleigh-Ritz step also come with some loss in other steps. The question is whether the loss is compensated or not.

The Spectrum Slicing method is described in algorithm 8 and in 9 when integrated in the self-consistent loop. Both algorithms calls some functions named Slices, Filter and Merge. These three functions correspond to the points that need to be worked on to make the slicing method work and that will be developed in the next tree sections.

Algorithm 8 Spectrum slicing eigensolver

$\Sigma \leftarrow \text{Slices}(H)$
for $(\sigma_i, \sigma_{i+1}, m_i) \in \Sigma$ **do**
  $f_i \leftarrow \text{Filter}(H, \sigma_i, \sigma_{i+1})$
  $X_i \leftarrow \text{RandomVectors}(N, m_i)$
  while X_i not converged **do**
    $X_i \leftarrow f_i(X_i)$
    $\Lambda_i, X_i \leftarrow \text{RayleighRitz}(X_i)$
  end while
end for
$\Lambda, X \leftarrow \text{Merge}(X_1, \ldots, X_{n_s}, \Lambda_1, \ldots, \Lambda_{n_s})$

Algorithm 9 Self-consistent field iterations using Spectrum Slicing

$X \leftarrow \text{RandomVectors}(N, m)$
$H \leftarrow \text{Hamiltonian}(X)$
while H not converged **do**
  $\Sigma \leftarrow \text{Slices}(H)$
  for $s_i \in \Sigma$ **do**
    $X_i \leftarrow \text{SelectVectors}(X, s_i)$
    $f_i \leftarrow \text{Filter}(H, s_i)$
    $X_i \leftarrow f_i(X_i)$
    $\lambda_i, X_i \leftarrow \text{RayleighRitz}(X_i)$
  end for
  $\lambda, X \leftarrow \text{Merge}(X_1, \ldots, X_{n_s})$
  $H \leftarrow \text{Hamiltonian}(X)$
end while

4.1. Determining the slices

To begin with, we will tackle in this section the challenges regarding the choice of the slices. For Spectrum Slicing to work well, the slices must be chosen such that :

- They cover entirely the part of the spectrum $[\lambda_1, \lambda_m]$ that we are interested in.
- The dimensions of the subspaces associated with each of the slices are close, so that the computational load is distributed evenly.
- The distance between two eigenvalues in different slices is large, to have good convergence rates (see section 3.3.2).
- The widths of the slices are as large and homogeneous as possible, as it is more difficult to filter a thin slice (see section 4.2).

4.1.1. Estimating the Density of states

In order to chose some appropriate slices, that satisfy the requirements above, one need an estimation of what the matrix eigenvalue spectrum looks like. Therefore, we will present here some method to approximate the Density of States (DOS).

Definition 4.1.1 (Density of states)**.** *The density of states of a matrix $A \in \mathbb{C}^{N\times N}$ with spectrum $\lambda_1, \dots, \lambda_N$ is defined as*

$$\phi(t) = \frac{1}{N}\sum_{j=1}^{N} \delta(t - \lambda_j) \tag{4.1}$$

where δ is the Dirac distribution.

Two common methods to estimate the DOS of a matrix are the Kernel Polynomial method (KPM) and the Lanczos method, that will be described next. Here we consider an hermitian matrix A with eigenvalues $\lambda_1, \dots, \lambda_N$ and some corresponding orthonormal eigenvectors $u_1, \dots, u_N$.

Kernel Polynomial Method

The idea of this method, described in [XLS18] and [LSY16] is to expand the Dirac δ-function in Chebyshev series (see appendix A.2). Chebyshev series expansions give approximation of function in $[-1, 1]$, so we assume, without loss of generality, that the eigenvalues $\lambda_1, \dots, \lambda_N$ of A are in $[-1, 1]$.

We define

$$\hat{\phi}(t) = \sqrt{1-t^2}\phi(t) = \sqrt{1-t^2}\frac{1}{N}\sum_{j=1}^{N} \delta(t - \lambda_j)$$

that will be expanded in Chebyshev series instead of ϕ for convenience. Then

$$\hat{\phi}(t) = \sum_{k=0}^{\infty} \mu_k T_k(t)$$

with the expansion coefficients

$$\mu_k = \frac{2-\delta_{k0}}{\pi}\int_{-1}^{1} \frac{1}{\sqrt{1-t^2}} T_k(t)\hat{\phi}(t) = \frac{2-\delta_{k0}}{n\pi}\sum_{j=1}^{N} T_k(\lambda_j).$$

The sum $\sum_{j=1}^{N} T_k(\lambda_j)$ is the trace of the matrix $T_k(A)$, which can be approximated (theorem 3.1 in [LSY16]) using a large number of vectors $v^1, \dots, v^{n_{\text{vec}}}$ whose component are obtained

from a normal distribution with zero mean and unit deviation and then normalized, with

$$T_k(A) \approx \frac{1}{n_{\text{vec}}} \sum_{l=1}^{n_{\text{vec}}} {v^l}^H T_k(A) v^l = \zeta_k.$$

The computation of the ${v^l}^H T_k(A) v^l$ can be done using the induction definition of Chebyshev polynomials, so that if the Chebyshev series expansion is truncated to M, a total of $M \times n_{\text{vec}}$ matrix vector multiplications are needed, that can be fully parellelized with n_{vec} process. The approximate DOS is then given by

$$\tilde{\phi}_M(t) = \frac{1}{\sqrt{1-t^2}} \sum_{k=0}^{M} \zeta_k T_k(t). \tag{4.2}$$

Lanczos Method

The Lanczos method is an iterative algorithm for determining the eigenvalues of a square matrix, which can therefore be used to estimate the DOS of a matrix. The method works by building iteratively an orthonormal basis $\{v_1, v_2, \ldots, v_M\}$ of the Krylov subspace $\text{Span}\{v_1, Av_1, \ldots, A^{M-1}v_1\}$ in a way that the matrix $T_M = {V_M}^H A V_M$, from the Rayleigh-Ritz procedure (with V_M the matrix with columns $v_1, \ldots, v_M$), is tridiagonal, that is

$$T_M = \begin{pmatrix} \alpha_1 & \beta_1 & & & \\ \beta_1 & \alpha_2 & \beta_2 & & \\ & \beta_2 & \ddots & \ddots & \\ & & \ddots & \ddots & \beta_{M-1} \\ & & & \beta_{M-1} & \alpha_M \end{pmatrix}$$

where the coefficients β_i and α_i are computed by induction (see [GVL13]). The tridiagonal matrix T_M can then be relatively easily diagonalized. The application of this method to compute the DOS is described in [XLS18] and [LSY16] also.

The Lanczos algorithm works so that the columns of V_M can be expressed as $v_k = p_{k-1}(A)v_1$, for $0 \leqslant k \leqslant M$ where $p_k \in \mathbb{C}_k[X]$, form a set of orthogonal polynomials with respect to the weighted spectral distribution

$$\phi_{v_1}(t) = \sum_{j=1}^{N} |\gamma_j|^2 \delta(t - \lambda_j)$$

where the coefficients $\gamma_1, \dots, \gamma_N$ are such that $v_1 = \sum_{j=1}^{N} \gamma_j u_j$. Indeed, for $1 \leqslant k_1, k_2 \leqslant M$,

$$\begin{aligned}
\langle p_{k_1}, p_{k_2} \rangle_{\phi_{v_1}} &= \int_{\mathbb{R}} \overline{p_{k_1}(t)} p_{k_2}(t) \phi_{v_1}(t) dt \\
&= \sum_{j=1}^{N} |\gamma_j|^2 \int_{\mathbb{R}} \overline{p_{k_1}(t)} p_{k_2}(t) \delta(t - \lambda_j) dt \\
&= \sum_{j=1}^{N} |\gamma_j|^2 \overline{p_{k_1}(\lambda_j)} p_{k_2}(\lambda_j) \\
&= \langle \sum_{j=1}^{N} \gamma_j p_{k_1}(\lambda_j) u_j, \sum_{j=1}^{N} \gamma_j p_{k_2}(\lambda_j) u_j \rangle_2 \\
&= \langle p_{k_1}(A) v_1, p_{k_2}(A) v_1 \rangle_2 \\
&= \langle v_{k_1}, v_{k_2} \rangle_2 \\
&= \delta_{k_1, k_2}
\end{aligned}$$

The polynomials p_k are generated with a 3 term recursion

$$p_{k+1}(X) = \frac{1}{\beta_{k+1}} \left(X p_k(X) - \alpha_k p_k(X) - \beta_k p_{k-1}(X) \right).$$

The Gaussian quadrature rule with this type of orthogonal polynomials was studied in [GW69] and yield the following approximation, for any function f,

$$\int_{\mathbb{R}} f(t) \phi_{v_1}(t) dt \approx \sum_{k=1}^{M} w_k f(\theta_k) = \int_{\mathbb{R}} f(t) \sum_{k=1}^{M} w_k \delta(t - \theta_k) dt$$

where (θ_k, y_k), $k = 1, \dots, M$, are the eigenpairs of T_M and $w_k = (\,1\ 0\ \dots\ 0\,) y_k$ is the first coordinate of y_k. So we will take the approximation

$$\hat{\phi}_{v_1}(t) = \sum_{k=1}^{M} w_k \delta(t - \theta_k).$$

It remains to choose v_1 so that $|\gamma_1|^2, \dots, |\gamma_N|^2$ are close to $\frac{1}{N}$. This is done again by choosing n_{vec} random vectors whose components are obtained from a normal distribution with zero mean and unit deviation and then normalized. This yields to the Lanczos DOS approximation

$$\tilde{\phi}_{v_1}(t) = \sum_{l=1}^{n_{\text{vec}}} \frac{N}{n_{\text{vec}}} \sum_{k=1}^{M} w_k^l \delta(t - \theta_k^l). \tag{4.3}$$

This method also require $M \times n_{\text{vec}}$ matrix vector products, that can be computed in parallel

with n_{vec} processes.

Some DOS estimations obtained with both methods are displayed in figure 4.1. In this few examples, we have observed better results with the Lanczos method, which is supported by the order of convergence of the two methods given in [XLS18] that are linear for KPM and quadratic for Lanczos. Morevover the KPM approximation is bad at both ends of the spectrum, and that is where we need a good approximation of the DOS. Therefore, in our algorithmic applications, we have always chosen the Lanczos approximation.

When applying the slicing method to constant matrices, we compute an estimated DOS at the very beginning and then choose some slice accordingly. However, when slicing is used in DFT, the matrix to diagonalize changes at each iteration, meaning that the estimated density of state might not remain correct all through the iterations and might need to be recomputed. When this happen, there is a third method that can be used which is simply to use the results of the previous iteration to compute the DOS.

4.1.2. Different strategies for choosing slices based on an estimated DOS

Once an estimation of density of states $\tilde{\phi}$ is known, it remains to use it to choose the slices. This can be done in various ways. The tree strategies that will be presented here where described in [WYBY20].

The first and most straightforward strategy would be to minimize the computational load discrepancy, by choosing some slices $[\sigma_1, \sigma_2], \ldots, [\sigma_{n_s}, \sigma_{n_s+1}]$ so that $\int_{\sigma_i}^{\sigma i+1} \tilde{\phi}(t)dt$ is a constant for all $i = 1, \ldots, n_s$. That is, we begin by estimating a lower bound $\lambda_{\min}$ of λ_1, with the power method for instance. Then we set $\sigma_1 = \lambda_{\min}$ and use numerical integration to find σ_2 such that $\int_{\sigma_1}^{\sigma_2} \tilde{\phi}(t)dt = \frac{m}{n_s}$ and repeat the process until we have n_s slices. This is described in algorithm 10.

Algorithm 10 Uniform weight strategy for choosing slices

$\lambda_{\min} \leftarrow$ LOWERBOUND(H)
$\sigma_1 \leftarrow \lambda_{\min}$
for $i = 1 \ldots n_s$ **do**
  $t \leftarrow \sigma_i$
  while $\int_{\sigma_i}^{t} \tilde{\phi}(\tau)d\tau < \frac{m}{n_s}$ **do**
    $t \leftarrow t + \Delta t$
  end while
  $\sigma_{i+1} \leftarrow t$
end for

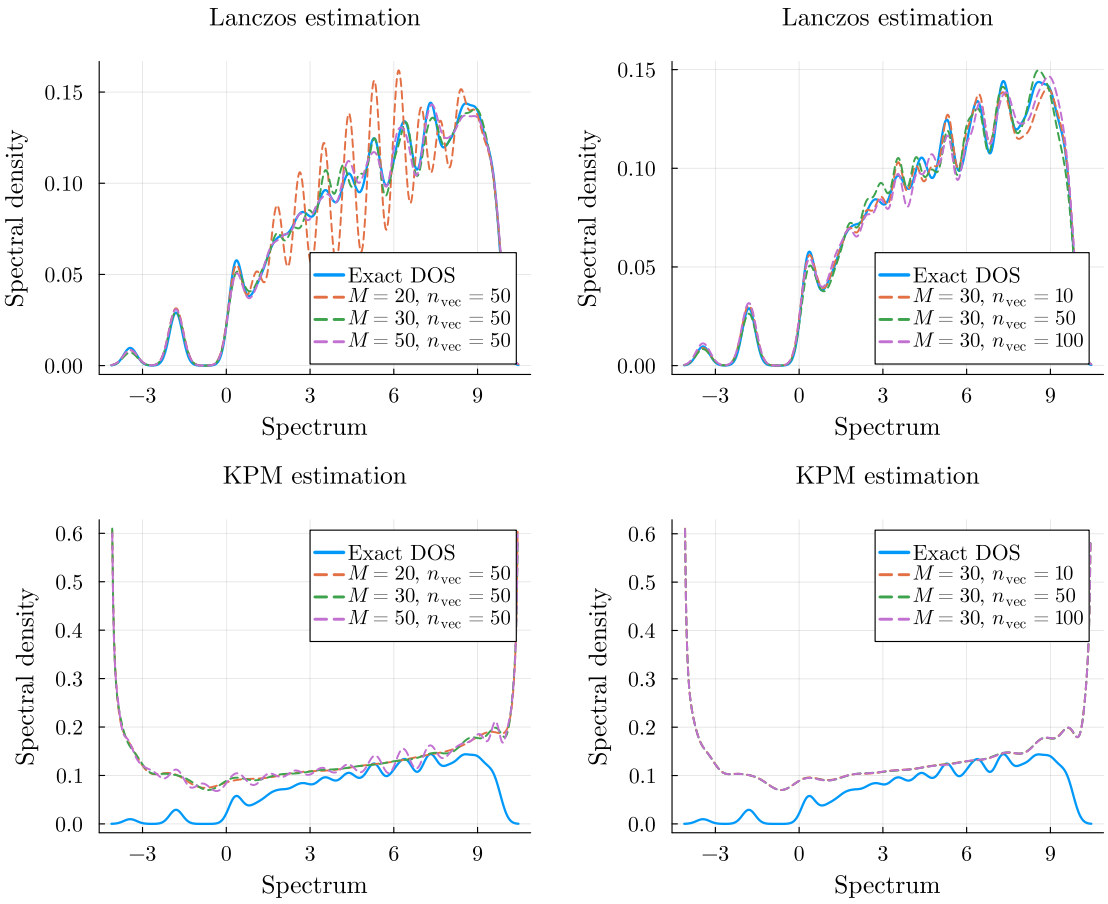


Figure 4.1.: Density of state estimation, computed with the Lanczos and kernel polynomial methods, for different values of M and n_{vec}, with Gaussian blurring, for a Hamiltonian H corresponding to a gold crystal.

The main drawback of this method is that the bounds of the slices σ_i can be chosen very close to the eigenvalues, which is very bad regarding the convergence.

A second possible strategy is to find the minima of the estimated DOS to place there the bounds of the slices. More precisely, we first estimate a lower bound $\lambda_{\min}$ of λ_1 (e.g. with the power method) and then find the last upper bound b that is such that $\int_{\lambda_{\min}}^{b} \tilde{\phi}(t)dt = m$. Then, we search for the local minima $m_1, m_3, \ldots, m_l$ of $\tilde{\phi}$ that are between $\lambda_{\min}$ and b. With $m_0 = \lambda_{\min}$ and $m_{l+1} = \sigma_{n_s+1}$, we now have $l+2$ slices. In case $l+2 \geqslant n_s$ i.e. we have to many slices, we compute the estimated number of eigenvalues (with multiplicity) in each slices with $\int_{m_i}^{m_{i+1}} \tilde{\phi}(t)dt$ and merge the two neighboring slices with the smallest combined number of eigenvalues, up until we are left with n_s slices. In case $l+2 < n_s$, we have chosen in our application to use only $l+2$ slices, because splitting a slice in two would likely lead

to having a slice boundary close to an eigenvalue. The strategy is described in algorithm 11.

Algorithm 11 Minima strategy for choosing slices

$\lambda_{\min} \leftarrow \textsc{LowerBound}(H)$
$t \leftarrow \lambda_{\min}$
while $\int_{\lambda_{\min}}^{t} \tilde{\phi}(\tau)d\tau < m$ **do**
  $t \leftarrow t + \Delta t$
end while
$b \leftarrow t$
$m_1, \ldots, m_l \leftarrow \textsc{Minima}(\tilde{\phi}, [\lambda_{\min}, b])$
$m_0 \leftarrow \lambda_{\min}$
$m_{l+1} \leftarrow b$
while $l + 2 > n_s$ **do**
  $i \leftarrow \textsc{ArgMin}(i \mapsto \int_{m_i}^{m_{i+2}} \tilde{\phi}(t)dt, [0, l-1])$
  $l \leftarrow l - 1$
  $m_{i+1}, \ldots, m_{l+1} \leftarrow m_{i+2}, \ldots, m_{l+2}$ ▷ $[m_i, m_{i+1}]$ and $[m_{i+1}, m_{i+2}]$ merged
end while
$\sigma_1 \ldots, \sigma_{l+2} \leftarrow m_0, \ldots, m_{l+1}$

This strategy is the best regarding the convergence rate as it yields well separated slices. The load balancing might be bad for this choice of slices, but we note that the strategy for merging slices in case we find to many minima is chosen to minimize load discrepancies. Also we might not have the amount of slices we wanted if not enough minima have been found. When using the Lanczos DOS estimation, one needs to use some Gaussian blurring. By playing with the width of the Gaussian, we can have more or less minima in the estimated DOS and thus improve load balancing and avoid cases where we don't have enough slices. However, if the Hamiltonian operator has many clustered eigenvalues, there is not much that can be done.

The last strategy requires to have some estimate of the eigenvectors, for instance from the previous iteration and consists in using a clustering algorithm (here the *k-means*) to compute clusters of eigenvalues that minimize the sum of distances in each of them and then choose the slice boundaries accordingly. This method will always give the wanted number of slices while avoiding placing close eigenvalues in two different slices. However the load balancing might be quite bad and if the spectrum don't have enough "natural" clusters, it could still have to put close eigenvalues in different clusters.

Figure 4.2 displays the slices obtained with these 3 different strategies for a density of states corresponding to the Hamiltonian of a gold crystal. We note that with the uniform

method and the k-means method, a slice boundary is placed very close to the second peak which is quite bad regarding convergence. Also with the minima strategy we have only been able to create 3 slices and not 5 like in the two other methods. Still its seems to be the best strategy as the eigenvalues are split into 3 dense clusters.

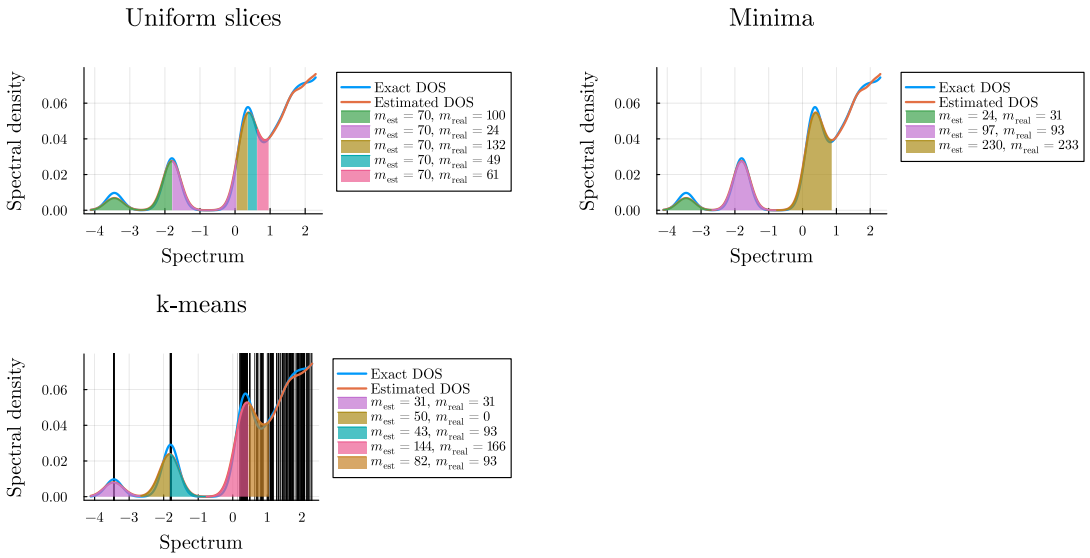


Figure 4.2.: Slices obtained using the three strategies presented in section 4.1.2, for the density of states obtained with a gold crystal, where the estimated DOS is obtained from the Lanczos method with $M = n_{\text{vec}} = 50$ for the uniform and minima strategies and with a Gaussian noise added to the eigenvalues for the k-means strategy.

4.1.3. Slices evolution

So far we have presented the DOS estimation and its use for choosing some slices. Now in this section we will tackle the filling of the slices and their evolution. Indeed, for the Spectrum Slicing method to work properly, we need the dimension of the estimated subspace of each slices to be greater or equal to the dimension of the actual eigensubspace corresponding to the slice. Otherwise, we might miss some eigenvalues and in the case of SCF iterations, this will completely ruin the convergence. Furthermore, in the self consistent field, the matrix H to diagonalize evolves. Therefore, we might need to adjust the slices as H changes.

In my implementation of the slicing method, the slices are determined as follow :

At the very first iteration, the DOS is estimated with the Lanczos method and then the slices are determined accordingly using one of the method above (section 4.1.2). In every

slice, we have an estimated number m_{est} of eigenvectors given by the DOS and we search for $m_{\text{est}} + n_{\text{extra}}$ vectors, where n_{extra} is a chosen parameter.

Then, for each slice $[\sigma_j, \sigma_{j+1}]$, we check if it is full. To do so, we check if one of the computed pair (μ_i, x_i) is an estimation of an eigenpair of H that is outside of the slice. For this check, we use lemma 4.1.1.

Lemma 4.1.1. *Let $H \in \mathbb{C}^{N\times N}$ a hermitian matrix and $x \in \mathbb{C}^N$ of norm 1. We denote $\mu = \frac{x^H H x}{x^H x} \in \mathbb{R}$ the Rayleigh quotient of x and $r = \|Hx - \mu x\|_2$ its residue. Then, there is an eigenvalue of H in $[\mu - r, \mu + r]$.*

Proof. As H is hermitian, we can write

$$H = U^H \Lambda U \tag{4.4}$$

with U a unitary matrix and $\Lambda \in \mathbb{R}^{N\times N}$ a diagonal matrix. Then,

$$\begin{aligned} r &= \|Hx - \mu x\|_2 \\ &= \|U^H \Lambda U x - \mu U^H U x\|_2 \\ &= \|U^H (\Lambda - \mu I_N) U x\|_2 \\ &\leqslant \|\Lambda - \mu I_N\|_2 = \min_{\lambda \in \mathcal{S}(H)} |\lambda - \mu| \end{aligned} \tag{4.5}$$

so $\lambda = \arg\min_{\lambda\in\mathcal{S}(H)} |\lambda - \mu|$ is in $[\mu - r, \mu + r]$. □

Thus, we check if $[\mu_i - r_i, \mu_i + r_i] \cap [\sigma_j, \sigma_{j+1}] = \emptyset$ where $r_i = \|Hx_i - \mu_i x_i\|_2$ is the residual of the estimated eigenpair (μ_i, x_i). If at least one estimated eigenpair is outside of the slice, we know that it is full. If the slice isn't full, we add some new random vectors at the next iteration. Precisely, we add $\lceil p \times m_i \rceil$ new vectors, where m_i is the current number of vectors in the slice and $p > 0$ is a chosen parameter.

In case the slicing method is applied to a constant matrix, this is enough to ensure that we will not miss any eigenpair.

In case the method is used in the self-consistent field, we also want to update the slices when the Hamiltonian matrix H changes. This is done as follows: Whenever we detect that a slice is not full, we increase a counter k_{up} and after several iterations, governed by a parameter n_{update} (see algorithm 12), the slices are updated. To update the slices, we use the current estimate of the eigenvalues, that gives us an approximation of the DOS, on which we apply one of the method above (section 4.1.2) to determine new slices. Then we need to allocate the estimated eigenvectors to their new slices. This correspond to the function SubspaceUpdate in algorithm 12, which is not detailed. The vectors are allocated to their slice using their Rayleigh quotient and in each slice we also add n_{extra}

vectors that should belong to the neighboring slices. These vectors are duplicated and should therefore be removed at one point but they are very useful to improve convergence and ensure that each slice is full.

This update is necessary to maintain load balancing between slices and ensure that the slices are well chosen all through the iterations. The update step isn't done at each step to avoid instabilities and to let time to each new eigenvector to converge in its slice before the slice is updated.

Algorithm 12 Slices evolution in the self-consistent loop

$H \leftarrow$ RandomHamiltonian(N)
$a_1, \ldots, a_{n_s+1} \leftarrow$ SlicesInit(H)
$X_1, \ldots, X_{n_s} \leftarrow$ SubspacesInit(H)
$k_{\text{up}} \leftarrow 0$
while H not converged **do**
  for $j = 1$ to n_s **do**
    $\mu_j, X_j \leftarrow$ EigenSolverSlice$(H, X_j, (a_j, a_{j+1}))$
    $m_j \leftarrow$ MissingVect$(H, \mu_j, X_j, (a_j, a_{j+1}))$
    $k_{\text{up}} \leftarrow k_{\text{up}} + m_1 + \ldots + m_{n_s}$
  end for
  $\mu, X \leftarrow$ Merge$(\mu_1, \ldots, \mu_{n_s}; X_1, \ldots, X_{n_s})$
  $H \leftarrow$ Hamiltonian(X)
  if $k_{\text{up}} \geqslant n_{\text{update}}$ **then**
    $a_1, \ldots, a_{n_s+1} \leftarrow$ SlicesUpdate(H, X)
    $X_1, \ldots, X_{n_s} \leftarrow$ SubspacesUpdate(H, X)
    $k_{\text{up}} \leftarrow 0$
  else
    $X_1, \ldots, X_{n_s} \leftarrow$ SubspacesAdd$(X_1, \ldots, X_{n_s}, m_1, \ldots, m_{n_s}, p)$ ▷ Adding new vectors in the slices that aren't full
    $k_{\text{up}} \leftarrow 2 \times k_{\text{up}}$
  end if
end while

4.2. Filters

Filtering is an essential part of the Spectrum Slicing method, and requires an important work to be able to filter a slice effectively and qualitatively. Indeed, filtering a slice, inside the spectrum, is much harder than filtering some of the extreme eigenvalues. For instance, using plain Chebyshev polynomials will not work here.

In this section, we describe different types of filters that have been used in the literature, separated in two categories, the polynomial filters and the rational filters.

Let $[a, b]$ with $a, b \in \mathbb{R}$ be the slice we want to filter.

4.2.1. Polynomial filters

To create a polynomial filter, we need a polynomial approximation of the characteristic function $\chi_{[a,b]}$ of the slice. Up to translation, we will assume here that our spectrum is contained in $[-1, 1]$.

Square filters

This first type of filter was presented in [SCS12] and consist in using the Chebyshev series expansion of the characteristic function $\chi_{[a,b]}$. That is, for $t \in [-1, 1]$,

$$\chi_{[a,b]}(t) = \sum_{i=0}^{\infty} \gamma_i T_i(t) \tag{4.6}$$

with

$$\gamma_i = \frac{2 - \delta_{i0}}{\pi} \int_{-1}^{1} \chi_{[a,b]}(t) T_i(t) \frac{1}{\sqrt{1 - t^2}} dt = \frac{2 - \delta_{i0}}{\pi} \int_{\arccos b}^{\arccos a} \cos(nt) dt$$

i.e.

$$\gamma_i = \begin{cases} \frac{1}{\pi} (\arccos a - \arccos b) & \text{if } i = 0 \\ \frac{2}{\pi} \frac{\sin(i \arccos a) - \sin(i \arccos b)}{i} & \text{if } i > 0. \end{cases} \tag{4.7}$$

Then, the polynomial filters of degree $k > 0$ are the truncated series with a damping coefficient, that is

$$P_k^{a,b} = \sum_{i=0}^{k} d_i^k \gamma_i T_i. \tag{4.8}$$

Here, d_i^k is the damping parameter that reduces oscillations (see appendix A.2). We will consider two types of damping: Jackson damping and Sigma damping i.e. $d_i^k = \sigma_i^k = \operatorname{sinc} i\theta_k$ with $\theta_k = \frac{\pi}{k+1}$ and also $d_i^k = 1$ when we don't want any damping. A few examples of Square filters are displayed in figure 4.3.

Dirac filters

The idea behind this type of filter is that we don't actually need our polynomial to be flat on $[a, b]$, but only to be steep in a and b. Therefore, instead of approximating $\chi_{[a,b]}$, one can only build bell-shaped filters. This can be done by using the Chebyshev series expansion of the Dirac δ-distribution centered in some $\gamma \in [a, b]$, denoted δ_γ, as described in the papers [LXV+15] and [LXES19].

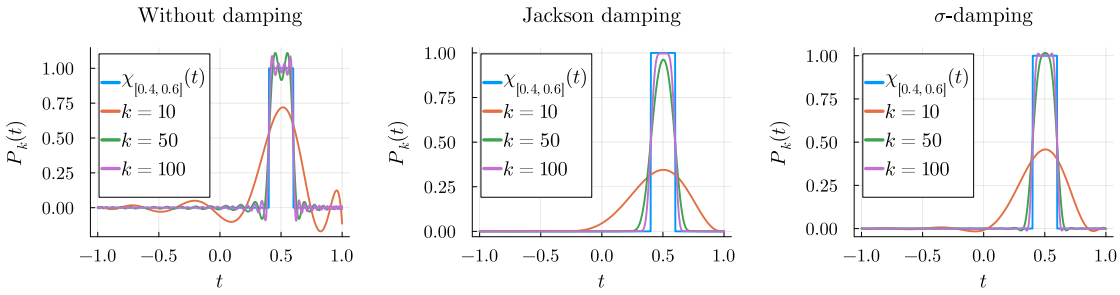


Figure 4.3.: Square filters of degree $k = 10$, 50 and 100, for the slice $[0.4, 0.6]$ with a spectrum contained in $[-1, 1]$, with no damping, Jackson damping and σ-damping.

Even though δ_γ is not a function but a distribution, the integral

$$\int_{-1}^{1} \delta_\gamma(t) T_i(t) \frac{1}{\sqrt{1-t^2}} dt = \frac{T_i(\gamma)}{\sqrt{1-\gamma^2}} = \frac{\cos(i \arccos \gamma)}{\sqrt{1-\gamma^2}}$$

can still be computed, which allows us to define the sum

$$P_k^\gamma = \sum_{i=0}^{k} \mu_k d_i^k T_k \tag{4.9}$$

with

$$\mu_k = \begin{cases} \frac{1}{2} & \text{if } k = 0 \\ \cos(i \arccos \gamma) & \text{else} \end{cases} \tag{4.10}$$

the Chebyshev expansion coefficients of δ_γ where the constant factors have been dropped and d_i^k, $i = 0, \ldots, k$ some damping coefficients. The width of P_n^γ depends of the degree k at which the Chebyshev series is truncated. The higher k, the "closer" P_k^γ is to δ_γ and therefore the thinner, as illustrated in figure 4.4.

Thus, the degree k is adjusted according to the width of the slice. For a given parameter $\tau \in]0, 1[$, k is chosen to be the smallest integer such that

$$\left| \frac{P_k^{\gamma_0}(a)}{P_k^{\gamma_0}(\gamma_0)} \right| < \tau \text{ and } \left| \frac{P_k^{\gamma_0}(b)}{P_k^{\gamma_0}(\gamma_0)} \right| < \tau$$

with $\gamma_0 = \frac{a+b}{2}$. Once k is fixed, we use a numerical optimization method to find γ such that $P_k^\gamma(a) = P_k^\gamma(b)$, so that the filter is well positioned and that there is no risk for a filtered eigenvalue outside of $[a, b]$ to be larger than one inside. The algorithm 13 describes this routine to determine k and γ.

The major benefit of this method is that the degree k is determined by the width of the

Algorithm 13

function ParamDirac(a, b, τ)
  $\gamma_0 \leftarrow \frac{a+b}{2}$
  $k = 0$
  while $\left|\frac{P_k^{\gamma_0}(a)}{P_k^{\gamma_0}(\gamma_0)}\right| < \tau$ and $\left|\frac{P_k^{\gamma_0}(b)}{P_k^{\gamma_0}(\gamma_0)}\right| < \tau$ **do**
    $k \leftarrow k + 1$
  end while
  $\gamma \leftarrow$ FindZero($\gamma \mapsto P_k^{\gamma_0}(b) - P_k^{\gamma_0}(a)$, $[a, b]$)
  return k, γ
end function

slice and not increased unnecessarily.

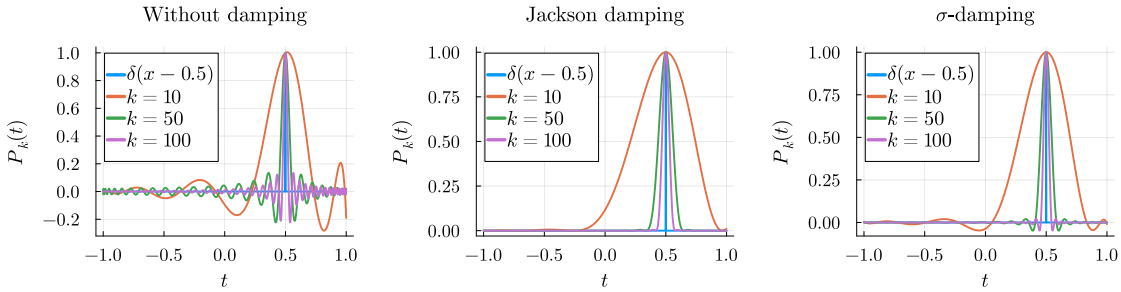


Figure 4.4.: Dirac filters of degree $k = 10$, 50 and 100, for the slice $[0.4, 0.6]$ with a spectrum contained in $[-1, 1]$, with no damping, Jackson damping and σ-damping.

With polynomial filters, the degree k is the number of times we have to apply the Hamiltonian operator to each vector. It is a costly operation, therefore we would like to do it as little as possible, meaning we want to maintain small degree. On figure 4.4 we see that the thinner the slice, the higher the degree needs to be. In figure 4.4, we have displayed Dirac filters with a degree up to 100 and the corresponding slice width is something we can expect to have with the Slicing method. This is the major drawback of the Spectrum Slicing method compared to Chebyshev filtering, where the polynomial degree usually remains under 10.

4.2.2. Rational filters

Now we will present a few rational filters that can be used to filter a slice interior to the spectrum. This filters can be very close to the characteristic function $\chi_{[a,b]}$ (e.g. *FEAST* [Pol09]) with a very low degree in the numerator and denominator. This makes them very good when the inverted matrix H^{-1} or its application to a vector is easily accessible, as it is the case with sparse matrices. In DFT, one has sparse discretized Hamiltonian when it

is discretized in real space, due to the local nature of the operators. In this case rational filters are highly suitable. However, in plane wave based DFT, the matrix H is dense and iterative method would be needed to compute the application of its inverse to a vector. This offsets the benefits given by lower degrees.

Shift-inverted filters

The first and most straightforward type of rational filter we will present here are *Shift-Inverted* filters. This filters are simple inversion i.e.

$$Q(t) = \frac{1}{t-c} \tag{4.11}$$

with $c = \frac{a+b}{2}$ the center of the slice. Then the quotient $\left|\frac{Q(\lambda_j)}{Q(\lambda_i)}\right| = \left|\frac{\lambda_i - c}{\lambda_j - c}\right|$ is always smaller than one when λ_i is in the slice $[a, b]$ and λ_j is outside of it. Thus the convergence rate will be smaller than one and convergence is ensured. This type of filters have been known and used for a long time, associated with the Lanczos method (see [KR91]). They are also studied in the more recent article [WYBY20]. Figure 4.5 represents a Shift-Inverted filter.

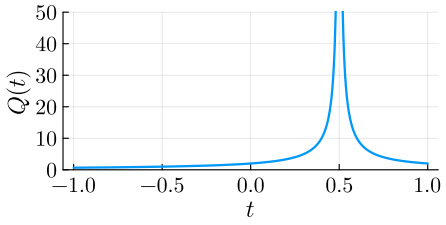


Figure 4.5.: Shift-inverted filter centered in 0,5.

FEAST

The method *FEAST* was first presented in [Pol09] and then in [PTP14]. The idea is to approximate the contour integral

$$\pi(t) = \frac{1}{2\pi i}\int_{\mathcal{C}} \frac{1}{z-t} dz,\ t \notin \mathcal{C}$$

where $\mathcal{C}$ is a circle centered in $c = \frac{a+b}{2}$ with radius $r = \frac{b-a}{2}$. From Cauchy's integral theorem, we know that $\pi(t) = 1$ when $\mathcal{C}$ circles t and $\pi(t) = 0$ else. Therefore, for $t \in \mathbb{R}$, $\pi(t) = \chi_{[a,b]}(t)$ is exactly the function we want to approximate. This integral is approximated using a quadrature rule with the parametrization $\varphi(\zeta) = c + re^{i\frac{\pi}{2}(1+\zeta)}$ for

$-1 \leqslant \zeta \leqslant 3$. This gives

$$\begin{aligned}
\pi(t) &= \frac{1}{2\pi i}\int_{-1}^{3} \frac{\varphi'(\zeta)}{\varphi(\zeta)-t} d\zeta \\
&= \frac{1}{2\pi i}\int_{-1}^{1} \frac{\varphi'(\zeta)}{\varphi(\zeta)-t} + \frac{\varphi'(2-\zeta)}{\varphi(2-\zeta)-t} d\zeta \\
&= \frac{1}{2\pi i}\int_{-1}^{1} \frac{ri\frac{\pi}{2}e^{i\frac{\pi}{2}(1+\zeta)}}{c+re^{i\frac{\pi}{2}(1+\zeta)}-t} - \frac{ri\frac{\pi}{2}e^{i\frac{\pi}{2}(3-\zeta)}}{c+re^{i\frac{\pi}{2}(3-\zeta)}-t} d\zeta \\
&= \frac{r}{4}\int_{-1}^{1} \frac{e^{i\frac{\pi}{2}(1+\zeta)}}{re^{i\frac{\pi}{2}(1+\zeta)}+c-t} - \frac{e^{-i\frac{\pi}{2}(1+\zeta)}}{re^{-i\frac{\pi}{2}(1+\zeta)}+c-t} d\zeta.
\end{aligned}$$

Then, we can use the Gauss-Legendre quadrature rule on $[-1,1]$ which gives

$$\pi_q(t) = \frac{r}{4}\sum_{i=1}^{q} w_i \left(\frac{e^{i\frac{\pi}{2}(1+\zeta_i)}}{re^{i\frac{\pi}{2}(1+\zeta_i)}+c-t} - \frac{e^{-i\frac{\pi}{2}(1+\zeta_i)}}{re^{-i\frac{\pi}{2}(1+\zeta_i)}+c-t} \right) \tag{4.12}$$

where $(w_i, \zeta_i)_{i=1\ldots q}$ are the weighs and nodes given by the Gauss-Legendre quadrature rule with q points (for some $q \geqslant 1$).

Some filters obtained with this method are displayed in figure 4.6. We see that this filters are very good for relatively small q. Also the q terms of the sum can be computed in parallel, which is why the FEAST method is said to have 3 levels of parallelization that are the slices, the vectors in each slice and the quadrature points in each vector. This method is used since 2013 as Intel-MKL's principal HPC eigensolver which illustrate its practical relevance.

Now as said above, when applied to dense matrices, we need to solve a linear system in order to apply the filter to a vector. In practice, this would be done inexactly with iterative solvers. The impact of this inexact inversion of the matrix was studied in [GP18].

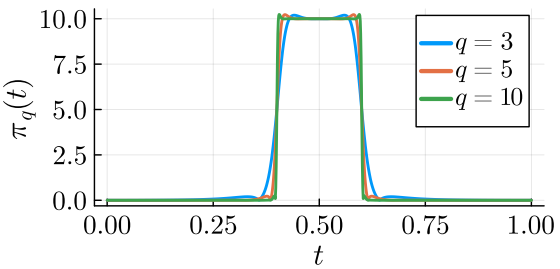


Figure 4.6.: FEAST filters with $q = 3$, 5 and 10 quadrature points, for the slice $[0.4, 0.6]$.

Least Square filter

The last type of rational filters we present here are called *Least Square* rational filters and where introduced in [XS16]. They are based on two observations: firstly it is not necessary that the filter is flat inside $[a, b]$ but only steep at its boundary and small on the rest of the spectrum. Seconddly some shifted linear systems $(H - \sigma I)y = x$ are easier to solve than other. For instance, the closer σ is to the real axis, the more iterations will be needed to solve the system with iterative methods. This is a downside of FEAST where the nodes obtained from the Gauss-Legendre quadrature tend to accumulate near the real axis. Therefore, in Least Square filters, we want to choose some poles $\sigma_1, \dots, \sigma_p$ in $\mathbb{C}$ that are advantageous for iterative linear solvers and then determine some coefficients $\alpha_1, \dots, \alpha_p$ so that

$$Q(t) = \sum_{i=1}^{p} \frac{\alpha_i}{t - \sigma_i} = \sum_{i=1}^{p} \alpha_i \varphi_i(t) \tag{4.13}$$

is a good filter (here we have denoted $\varphi_i(t) = \frac{1}{t-\sigma_i}$). In this section, we assume our slice to be centered in 0 with radius r and the spectrum to be in $[-1, 1]$.

Now to measure the quality of a filter, we introduce the scalar product $\langle \cdot, \cdot \rangle_w$ defined by $\langle f, g \rangle_w = \int_{-\infty}^{+\infty} f(t)\overline{g(t)}w(t)dt$ with

$$w(t) = \begin{cases} 0 & \text{if } |t| > 1 \\ \beta & \text{if } |t| < r \\ 1 & \text{else} \end{cases}$$

where $0 < \beta < 1$ is small. Then, we want to find the coefficients $\alpha_1, \dots, \alpha_p$ that minimize

$$\|Q - \chi_{[-r,r]}\|_w.$$

By using the weight function w, we can reduce the importance of the shape of our filter between $-r$ and r while ensuring that it remains small on the rest of the spectrum. The optimization problem is equivalent to

$$\min_{\alpha \in \mathbb{C}^p} \alpha^H G \alpha - \eta^H \alpha - \alpha^H \eta$$

where $G = (\langle \varphi_i, \varphi_j \rangle_w)_{1 \leqslant i,j \leqslant p}$ is a positive definite hermitian matrix in $\mathbb{C}^{p \times p}$, η is a vector in $\mathbb{C}^p$ with $\eta^H = \big(\langle \chi_{[-r,r]}, \varphi_1 \rangle \dots \langle \chi_{[-r,r]}, \varphi_p \rangle\big)$ and $\alpha = (\alpha_1 \dots \alpha_p)^T$ is the vector of sought

coefficients. The optimal solution of this convex quadratic minimization problem is

$$\alpha = 2G^{-1}\eta \tag{4.14}$$

which can be computed numerically.

Some example of least square filters are displayed in figure 4.7. We note that these filters are of good quality, with only 3 or 5 poles. Moreover, their quality improves when the poles are chosen closer to the real axis.

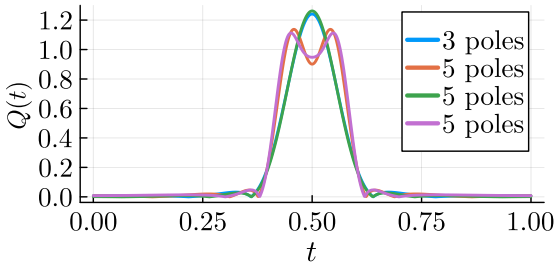


Figure 4.7.: Least-square filters obtained with 4 different sets of poles, for the slice $[0.4, 0.6]$.

4.3. Merging results between slices

Now that we know how to choose some slices and some filters to apply the filtering method, we want to know how to put together the results obtained independently on each slices. If we had an ideal algorithm giving us exactly the eigenpairs whose eigenvalues are in a given slice, the entire set of sought eigenpairs would simply be the reunions of the ones found in each slice. However, our algorithm is not perfect and taking the reunion of all computed eigenpairs would give us a lot of duplicates. For instance, as described in section 4.1.3, we need to compute more vectors than their are in a slice, to ensure that it is full and we are not missing any. Therefore, we need a routine to identify duplicates between slices.

In most cases, using the residual $r = \|Hx - \mu x\|$ of the eigenpair (μ, x) is enough to determine whether it is inside the slice $[a, b]$ in which it was computed or not. Indeed, using lemma 4.1.1, we know that the eigenvalue λ of H approximated by μ is in $[\mu - r, \mu + r]$. Thus, if $[\mu - r, \mu + r] \cap [a, b] = \emptyset$ we know that the eigenpair can be discarded because it does not belong to the slice $[a, b]$ on the contrary, if $[\mu - r, \mu + r] \subseteq [a, b]$ the eigenpair is kept. Now in the last case where $[\mu - r, \mu + r]$ is neither outside $[a, b]$ nor inside, a doubt remains. This can happen for eigenpairs that are not yet well converged or very close to a slice boundary.

In case where the residual is not sufficient to make a decision, we will use the method described in [SCS12] that compares principle angles between neighboring slices to identify duplicates. Let $[a, b]$ and $[b, c]$ be two neighboring slices, $(\mu_1, x_1), \ldots, (\mu_l, x_l)$ the estimated eigenpairs computed on slice $[a, b]$ such that, for $i = 1 \ldots l$, $[\mu_i - r_i, \mu_i + r_i] \cap [a, b] \neq \emptyset$ and $[\mu_i - r_i, \mu_i + r_i] \cap [b, c] \neq \emptyset$ with $r = \|Hx_i - \mu_i x_i\|$ and $y_1, \ldots, y_{m_s}$ the estimated eigenvectors computed on $[a, b]$ without the ones discarded from their residual. We want to identify the vectors in $x_1, \ldots, x_l$ that have duplicates in $y_1, \ldots, y_{m_s}$. However, since eigenvectors are not unique, it can not be done by directly comparing the vectors one by one, rather, we will check whether the two associated subspaces are orthogonal or not. We denote $Q = (x_1 | \ldots | x_l)$ and $R = (y_1 | \ldots | y_{m_s})$. Let

$$Q^H R = U \Sigma V^H$$

be the singular value decomposition of $Q^H R$. The diagonal of Σ is the cosines of the principal angles between the two subspaces associated to Q and R ([BG73]) i.e.

$$\Sigma = \begin{pmatrix} \cos\theta_1 & & \\ & \ddots & \\ & & \cos\theta_n \end{pmatrix}$$

with $n = \max(l, m_s)$ and $\theta_1, \ldots, \theta_n$ the principal angles. Then, the number k of singular values of Σ close to one is the number of duplicate vectors and it remains to find to which column of Q they correspond. We denote $n_1, \ldots, n_k$ the indices of the singular values close to one, then for $j = 1, \ldots, k$, the n_j-th column U_{n_j} of U gives the coefficients of the linear combination of $x_1, \ldots, x_l$ whose angle with an element of $\mathrm{Span}\{y_1, \ldots, y_{m_s}\}$ is θ_{n_j}. Since we might have duplicate eigenvalues, we cannot simply examine the columns of U for their maximum element because this could give us the same vector for different columns, as we cannot expect each angle to be primarily associated with only one vector. Therefore, we define, for $i = 1 \ldots l$,

$$m_i = \max_{1 \leqslant j \leqslant k} |(U_{n_j})_i| \tag{4.15}$$

the maximal contribution of the vector x_i in the angles $\theta_{i_1}, \ldots, \theta_{i_k}$. Then, the duplicates vectors are the ones that contribute the most, that is $x_{i_1}, \ldots, x_{i_k}$ when $m_{i_1} \geqslant m_{i_2} \geqslant \ldots \geqslant m_{i_l}$.

When H is a constant matrix, we have to merge the results obtained in different slices only once, when the algorithm has converged on every slice (see algorithm 8). Therefore,

we can expect our estimated eigenpairs to have small residual and if our slices where well chosen, the residual check should be enough to merge the slices. In DFT however, we have to recalculate the matrix H at each step and to do so, we need to merge the vectors obtained from each slice at each step (see algorithm 9). When the slices are not yet well converged, we might need to compare principal angles. This is a crucial step because having duplicate eigenpairs will prevent the Hamiltonian operator from converging.

5. Comparison of time efficiency in the Slicing method and Chebyshev filtering method

In this section we will study the time complexity of Spectrum Slicing and compare it to that of Chebyshev filtering. The objective is to determine if and when the Slicing method becomes profitable compared to Chebychev filtering. As explained above, Slicing is designed to speed up the Rayleigh-Ritz stage. However, this involves more complex and therefore more costly filters. We therefore want to know if the gain on the Rayleigh-Ritz compensates for the loss on the filtering step.

5.1. Study of Chebyshev filtering and Spectrum Slicing on a simplified model

To compare the two methods, we want to choose the degree of the polynomials corresponding to the same rate of convergence in both methods. The next two sections will explicit the relation between the degree of the polynomials and the convergence rate for Chebyshev filtering and Spectrum slicing.

In order to study this dependence, we consider a model system where we want to compute the m smallest eigenvalues of a $N \times N$ hermitian matrix H whose eigenvalues $\lambda_1, \ldots, \lambda_N$ (sorted in increasing order) are evenly distributed in the interval $[-1, 1]$ (i.e. $\lambda_i = -1 + \frac{2i-2}{N-1}$,, for $1 \leqslant i \leqslant N$). We denote $u_1, \ldots, u_N$ some corresponding orthonormal eigenvectors and $\mathcal{U} = \mathrm{Span}\{u_1, \ldots, u_m\}$ the eigensubspace corresponding to the m smallest eigenvalues.

5.1.1. Chebyshev filtering

In the Chebyshev-filtered Subspace iteration method, the polynomials

$$P_k(X) = T_k\left(\frac{X - c}{r}\right) \tag{5.1}$$

are used to filter $\mathcal{U}$, with $c = \frac{1}{2}(\lambda_{m+1} + \lambda_N) = m/(N-1)$ and $r = \frac{1}{2}(\lambda_{N+1} - \lambda_{m+1}) = 1 - m/(N-1)$ assuming distributed eigenvalues in $[-1, 1]$. The convergence properties of the subspace iteration method was presented in section 3.3.2. Here the convergence rate of the method as a whole corresponds to the rate of convergence of the eigenvalue that converges the slowest, that is λ_m. Hence the convergence rate of Chebyshev filtering is given by

$$\mu = \left| \frac{P_k(\lambda_{m+1})}{P_k(\lambda_m)} \right| = \left| \frac{T_k(-1)}{T_k(-1 - \frac{2}{(N-1)r})} \right|.$$

Using lemma A.1.3 and A.1.4, we have $T_k'(-1) = (-1)^{k-1}k^2$ so for large N we can approximate T_k around -1 with a series expansion,

$$T_k(-1 - \frac{2}{(N-1)r}) = (-1)^k - (-1)^{k-1}k^2\frac{2}{Nr} + \mathcal{O}\left(\frac{1}{N^2}\right) = (-1)^k\left(1 + \frac{2k^2}{Nr}\right) + \mathcal{O}\left(\frac{1}{N^2}\right)$$

which gives

$$\mu = 1 - \frac{2k^2}{N(1 - m/N)} + \mathcal{O}\left(\frac{1}{N^2}\right). \tag{5.2}$$

5.1.2. Spectrum slicing

For Spectrum Slicing, we will consider δ-Dirac filters with σ-damping, that is

$$P_k^\gamma(X) = \sum_{i=0}^{k} \mu_i \sigma_{i,k} T_i(X) \tag{5.3}$$

where

$$\mu_i = i\cos(i\arccos\gamma) \text{ and } \sigma_{i,k} = \begin{cases} 1 & \text{if } i = 0 \\ \frac{\sin(i\frac{\pi}{k+1})}{i\frac{\pi}{k+1}} & \text{else} \end{cases} \quad \text{for } i, k \geqslant 0. \tag{5.4}$$

We will call n_s the number of slices which we will consider to be evenly distributed on $[\lambda_1, \lambda_m]$ such that each slice contains about $m_s = \frac{m}{n_s}$ eigenvalues.

We first consider a slice centered around 0 and containing m_s eigenvalues. The last eigenvalue in the slice is around $\frac{m_s}{N} = \frac{m}{Nn_s}$ and the first one outside is around $\frac{m_s}{N} + \frac{2}{N}$. Then an approximate convergence rate μ is given by

$$\mu = \left| \frac{P_k^0(\frac{m_s}{N} + \frac{2}{N})}{P_k^0(\frac{m_s}{N})} \right|$$

which can be approximated via the series expansion

$$P_k^0(\frac{m_s}{N}+\frac{2}{N}) = P_k^0(\frac{m_s}{N}) + P_k^{0\prime}(\frac{m_s}{N})\frac{2}{N} + \mathcal{O}\left(\frac{1}{N^2}\right).$$

Then

$$\mu = \left|\frac{P_k^0(\frac{m_s}{N}) + P_k^{0\prime}(\frac{m_s}{N})\frac{2}{N} + \mathcal{O}\left(\frac{1}{N^2}\right)}{P_k^0(\frac{m_s}{N})}\right| = 1 + \frac{P_k^{0\prime}(\frac{m_s}{N})}{P_k^0(\frac{m_s}{N})}\frac{2}{N} + \mathcal{O}\left(\frac{1}{N^2}\right) = 1 - p_k^0\frac{2}{N} + \mathcal{O}\left(\frac{1}{N^2}\right)$$

where

$$p_k^0 = -\frac{P_k^{0\prime}(\frac{m_s}{N})}{P_k^0(\frac{m_s}{N})}$$

is positive.

We will compute p_k^0 numerically. For a given slice of half-width l_s, the degree is chosen as explained in section 4.2.1, that is such that $P_k^0(l_s) = \tau P_k^0(0)$ for a given τ in $]0,1[$. Figure 5.1 shows the value l_s for which $P_k^0(l_s) = \tau P_k^0(0)$, with respect to k, for different values of τ. We observe that for high degrees, l_s evolves as $\frac{1}{k}$. We write $l_s \approx \frac{\alpha_\tau}{k}$. In our model with evenly distributed eigenvalues, this means that the degree needed to filter (circa) m_s eigenvalues, in a slice centered around 0, can be approximated with

$$k = \frac{\alpha_\tau N}{m_s} = \frac{\alpha_\tau N n_s}{m}. \tag{5.5}$$

The degree varies linearly with the number of slices and inversely with the ratio $\frac{m}{N}$ of sought eigenpairs.

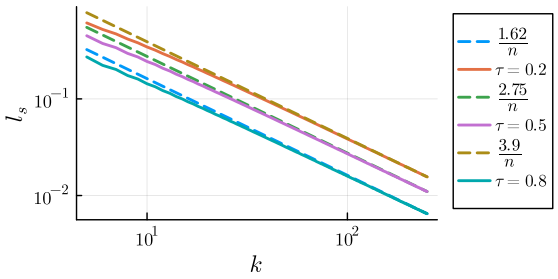


Figure 5.1.: Values of l_s such that $P_k^0(l_s) = \tau P_k^0(0)$ with respect to the degree k, for different values of τ.

Figure 5.2 shows the value of $p_k^0 = |P_k^{0\prime}(l_s)/P_k^0(l_s)|$ where l_s is such that $P_k^0(l_s) = \tau P_k^0(0)$. It seems that p_k has a quasi-linear dependence to the degree k. Hence we write $p_k \approx \beta_\tau k$.

Then the convergence rate μ can be estimated with

$$\mu = 1 + p_k \frac{2}{N} + \mathcal{O}\left(\frac{1}{N^2}\right) \approx 1 + \beta_\tau \alpha_\tau \frac{2n_s}{m} + \mathcal{O}\left(\frac{1}{N^2}\right). \tag{5.6}$$

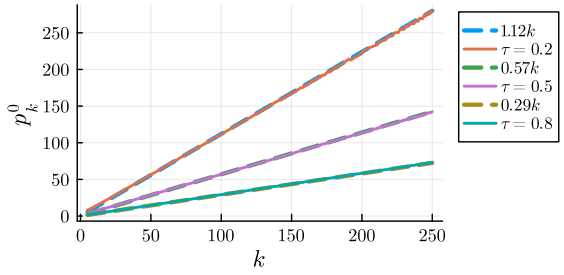


Figure 5.2.: Value of p_k^0 with respect to the degree k for different τ.

The hypothesis that the slice is centered in 0 is actually disadvantageous since this is where the δ-Dirac filters are the widest (see figure 5.3). Therefore, we have computed the width of the P_k^γ for various γ and found that it sill evolves as $\frac{1}{k}$, only the prefactor α_τ changes. The same goes for the values of $p_k^\gamma = |P_k^{\gamma\prime}(\gamma - l_s)/P_k^\gamma(\gamma - l_s)|$ with l_s such that $P_k^\gamma(\gamma - l_s) = \tau P_k^\gamma(\gamma)$. Figure 5.4 shows the values of α_τ and β_τ with respect to γ. Based on these graphs, we will consider that the convergence rate, that depends on $\alpha_\tau \beta_\tau$, does not depend on the position of our slices for a given τ. However the factor α_τ for the degree depends on γ. In our model, the center of the slices that is the closest to 0, i.e. the one for which we will need the higher polynomial degree, can be estimated by $\gamma = \min(2m/N - 1, 0)$. Therefore we add an upper index m/N to the prefactors α and β to express their dependency to the proportion of sought eigenpairs. This way, the estimated polynomial degree needed to filter m eigenvalues with n_s slices can be rewritten as

$$k = \alpha_\tau^{m/N} n_s \frac{N}{m}. \tag{5.7}$$

The values $\alpha_\tau^{m/N}$ and $\beta_\tau^{m/N}$ are computed numerically to be used in the rest of the calculations.

5.2. Comparison of polynomial degrees needed in both methods

Now that we know how the polynomial degree and the convergence rate are related in the two methods, we will compare the degree needed to have the same convergence rate μ with the Spectrum Slicing method and Chebyshev filtering method.

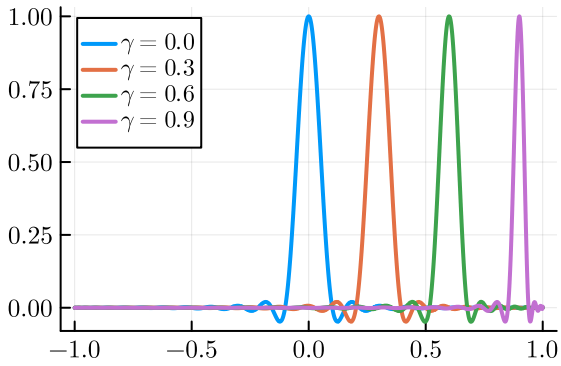


Figure 5.3.: Dirac filters of degree 50 on $[-1, 1]$, with σ-damping, centered in different γ.

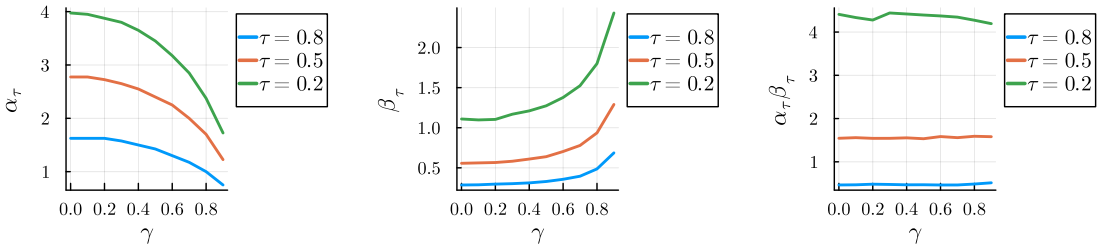


Figure 5.4.: Factors α_τ, β_τ and product $\alpha_\tau\beta_\tau$ with respect to the position of the slice γ.

For a given degree k_c of the Chebyshev filter, the convergence rate will be $\mu = 1 - 2{k_c}^2/(N(1 - m/N))$ and, for the Spectrum Slicing method to have the same convergence rate, we will need δ-Dirac filters of degree

$$k_s = \frac{{k_c}^2}{\beta_\tau^{m/N}(1 - m/N)} \tag{5.8}$$

Then since the width of the slices is determined by the degree (up to τ) this also determines the number of slices

$$n_s = \frac{m}{N}\frac{k_s}{\alpha_\tau^{m/N}} = \frac{m}{N}\frac{{k_c}^2}{\alpha_\tau\beta_\tau(1 - m/N)}. \tag{5.9}$$

Thus, the degree and number of slices in the slicing method are both multiples of k_c^2.

We want the number of slices to be high and the degree to be small. The quotient of the two is $n_s/k_s = (m/N) \times 1/\alpha_\tau^{m/N}$ which grows with m/N (see figure 5.5) so Spectrum Slicing will work best for a high proportion of sought eigenpairs. Figure 5.6 shows the evolution of k_s and k_c with respect to the number of slices (the convergence rate and therefore k_s and k_c is determined by the width of the slices), for a few proportions of sought eigenpairs, this

illustrate again that wanting only a small proportion of eigenpairs is penalizing and that the difference in degree between slicing and Chebyshev filtering is very significant.

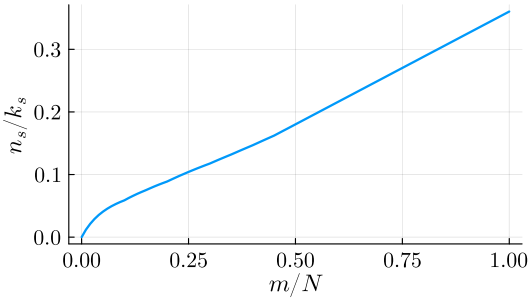


Figure 5.5.: Quotient n_s/k_s for $\tau = 0,5$ with respect to m/N the proportion of sought eigenpairs.

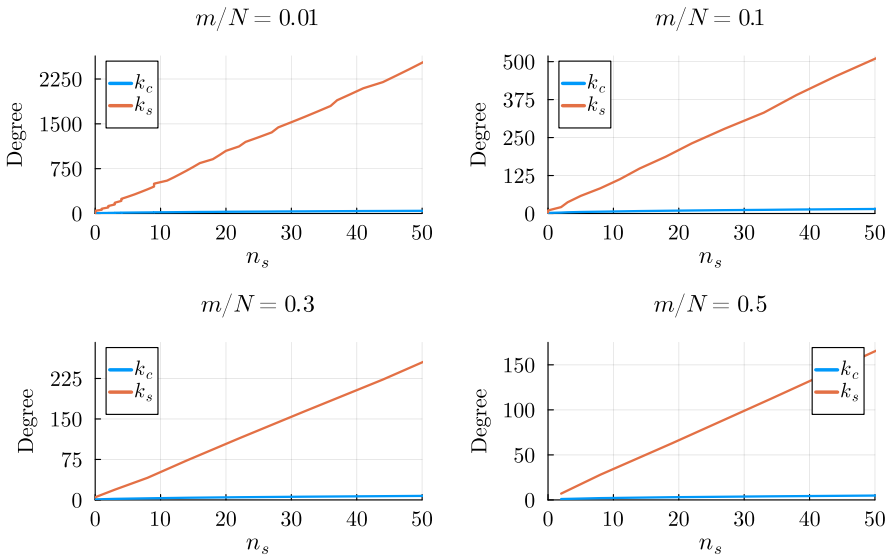


Figure 5.6.: Degree k_s required in the Spectrum Slicing method ($\tau = 0,5$) with respect to the number of slices n_s and degree k_c required in Chebyshev filtering method for the same convergence rate, for different proportions of sought eigenpairs m/N.

5.3. Time complexity of the different steps involved in the subspace iteration method

We want to compare the Chebyshev filtering and Spectrum Slicing techniques. On the one hand, Chebyshev filtering only requires a low degree filter, so few applications of the Hamiltonian operator H, but then performs the Rayleigh-Ritz step in a m dimensional space. On the other hand, the slicing method needs higher degree filters but then apply the Rayleigh-Ritz step in smaller dimension. To be able to compare the two methods, we need to estimate the time required for an application of the Hamiltonian to a vector and for the Rayleigh-Ritz method. This is what we will focus on in this section.

For our analysis, we will use some time analysis that have been made on a GPU based supercomputer with the Chebyshev filtering method, presented in table 5.1. Several DFT calculations where made on 3 test cases named Ti255-A, Ti255-B and Ga2O3-A. Ti255-A and Ti255-B are titanium crystals and Ga2O3-A is a gallium (III) oxide. For each of these calculations, the total time spent in the application of the Hamiltonian H, the application of B^{-1} and the Rayleigh-Ritz procedure where measured and correspond to the columns T_H^{tot}, $T_{B^{-1}}^{\text{tot}}$ and T_{RR}^{tot} respectively. These calculations are made in the PAW formalism (see section 2.5) so the eigenvalue problem to solve is actually a generalized eigenvalue problem $HX = BX\Lambda$, where B is called the overlap matrix. It can be rewritten $B^{-1}HX = X\Lambda$. This is where the need of applying a matrix B^{-1} comes from. The table also contains some parameters related to the computed physical systems. N is the number of plane wave used to discretized the Hamiltonian (hence the size of the matrix H), N_{at} is the number of atoms considered, N_p is the number of projectors used to write the pseudopotentials, m is the number of sought bands (i.e. eigenpairs), k_c is the degree of the Chebyshev polynomial used and n_{step} is the number of self-consistent steps. Finally, b_{pp} stands for "band per processor" and is the number of eigenpairs that where handled by each processor.

Using these data, we fill the table 5.2 in which T_H and $T_{B^{-1}}$ are the average time needed for *one* application of H and B^{-1} to *one* vector and T_{RR} is the average time needed for *one* Rayleigh-Ritz step. T_H and $T_{B^{-1}}$ are obtained by dividing the total time by k_c, n_{step} and b_{pp} and T_{RR} by dividing the total time with the number of steps only since it does not depend on the number of processor used.

Table 5.2 will be used to give some computation time estimation of the application of H and B^{-1} and the Rayleigh-Ritz step, depending on N, N_p and m. This computation times are related to the material used and the implementation but since we are mostly interested in the relation between them it is sufficient.

	b_{pp}	N	N_{at}	N_p	m	k_c	n_{step}	T_H^{tot} (s)	$T_{B^{-1}}^{\text{tot}}$ (s)	$T_{\text{RR}}^{\text{tot}}$ (s)
Ti255-A	256	17092	255	4590	2048	4	17	4,002	7,944	6,31
Ti255-A	128	17092	255	4590	2048	4	17	2,2	4,608	5,938
Ti255-B	512	34590	255	4590	4096	4	22	20,132	21,527	35,325
Ti255-B	256	34590	255	4590	4096	4	22	10,386	11,201	32,693
Ga2O3-A	720	77396	1960	8640	8640	4	4	59,881	196,31	55,684
Ga2O3-A	360	77396	1960	8640	8640	4	4	32,852	96,91	50,055

Table 5.1.: Total execution time of the application of the Hamiltonian, the application of B^{-1} and the Rayleigh-Ritz method, measured on a GPU based computer for DFT simulations of different physical systems with their size parameters (N, N_{at}, N_p and m), the degree k_c of the Chebyshev polynomial used, the number of SCF steps n_{step} and the number band per processor b_{pp}.

	N	N_{at}	N_p	m	T_H (ms)	$T_{B^{-1}}$ (ms)	T_{RR} (s)
Ti255-A	17092	255	4590	2048	0,2299	0,4563	0,3711
Ti255-A	17092	255	4590	2048	0,2528	0,5294	0,3493
Ti255-B	34590	255	4590	4096	0,4468	0,4778	1,606
Ti255-B	34590	255	4590	4096	0,4610	0,4972	1,486
Ga2O3-A	77396	1960	8640	8640	5,198	17,04	13,92
Ga2O3-A	77396	1960	8640	8640	5,703	16,82	12,51

Table 5.2.: Average execution time of one application of the Hamiltonian, one application of B^{-1} and one execution of the Rayleigh-Ritz method, measured on a GPU based computer for DFT simulation of different physical systems with their size parameters (N, N_{at}, N_p and m), computed from table 5.1.

5.3.1. Application of the Hamiltonian to a vector

To analyse the time complexity of the application of the Hamiltonian operator, we follow the work of [LT15].

When using pseudopotentials the Hamiltonian operator can be expressed as

$$H = T_s + V_{\text{loc}} + V_{\text{nonloc}}. \tag{5.10}$$

The kinetic energy operator is a diagonal matrix in the plane wave basis, so its application is in $\mathcal{O}(N)$. The local potential operator is a multiplication in real space so its application to a

vector can be computed using the inverse and direct Fourier transform of the vector. Using FFT, its application is in $\mathcal{O}(N\log(N))$. Finally, the non-local potential operator depends on the atomic data and can be expressed as $V_{\text{nonloc}} = PD_VP^H$ where P is a $N \times N_p$ matrix containing the projectors used to model the core electrons and $D_V \in \mathbb{C}^{N_p \times N_p}$ is block diagonal. Thus the application of V_{nonloc} is in $\mathcal{O}(NN_p + {N_p}^2)$.

Figure 5.7 shows the time spent for one application of the Hamiltonian with respect to NN_p for the data of table 5.2. we observe that the computation time seems to grow linearly with respect to NN_p hence we will ignore the $\mathcal{O}(N\log N + {N_p}^2)$ term and estimate

$$T_H(N, N_p) \approx ANN_p \tag{5.11}$$

with $A = 3 \times 10^{-12}$ s.

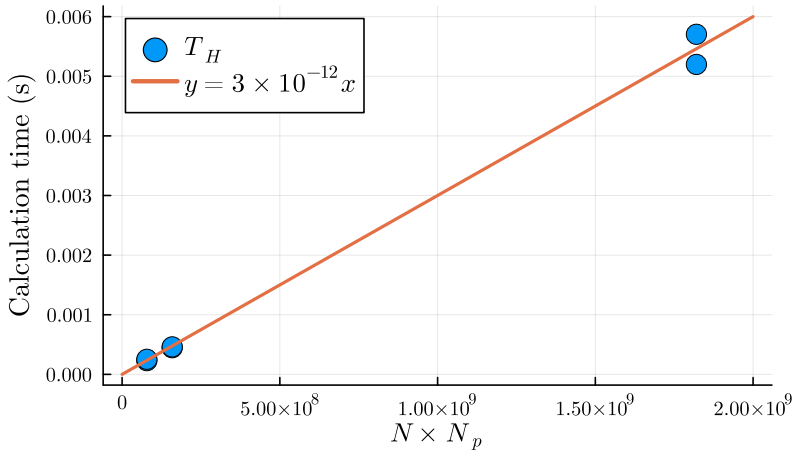


Figure 5.7.: Measured calculating time for one application of the Hamiltonian operator to a vector on GPU calculators with respect to the product NN_p.

5.3.2. Inversion of the overlap matrix

The overlap matrix B appears when using the PAW formalism for pseudopotentials. The operator to diagonalize is then $B^{-1}H$ and B has the form

$$B = I + PD_BP^H \tag{5.12}$$

where P is the same $N \times N_p$ projector matrix as above and $D_B \in \mathbb{C}^{N_p \times N_p}$ is block diagonal. Then B can be inverted using the Woodbury formula [Woo50] that yields

$$B^{-1} = I + P({D_B}^{-1} + P^HP)^{-1}P^H. \tag{5.13}$$

Thus we now only need to inverse an $N_p \times N_p$ matrix. This inversion can be done in $\mathcal{O}(N_p^2)$ as explained in [LT15] and corroborated by our data. We then use table 5.2 to estimate the time prefactor. This gives

$$T_{B^{-1}}(N_p) \approx C{N_p}^2 \tag{5.14}$$

with $C = 3 \times 10^{-11}$ s (see figure 5.8).

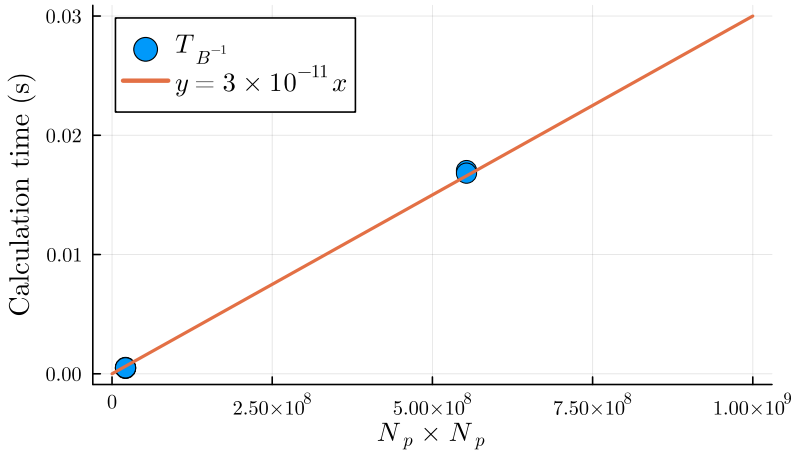


Figure 5.8.: Measured calculating time for one application of the inverted overlap matrix to a vector on GPU calculators with respect to the number of projector N_p.

5.3.3. Rayleigh-Ritz method

For a matrix $H \in \mathbb{C}^{N \times N}$ and a matrix $X \in \mathbb{C}^{N \times m}$ with orthonormal columns, the Rayleigh-Ritz method consist of 3 steps :

1. Compute the $m \times m$ matrix $A = X^H H X$
2. Solve the eigenvalue problem $AY = Y\Lambda$
3. Compute the Ritz vectors $Z = XY$

to which we sometimes add the orthonormalization step of the matrix X. Here the most time consuming step is the diagonalization of the matrix A, since matrix products are accelerated by parallelization. We have to use an iterative algorithm, which scale in the general hermitian case in $\mathcal{O}(m^3)$. We estimate the timing scaling factor using table 5.2. Figure 5.9 shows the time spent in the Rayleigh-Ritz step as a function of the subspace dimension. From it, we estimate that the time per Rayleigh-Ritz step is

$$T_{\mathrm{RR}}(m) \approx Bm^3 \tag{5.15}$$

with $B = 2 \times 10^{-11}$ s.

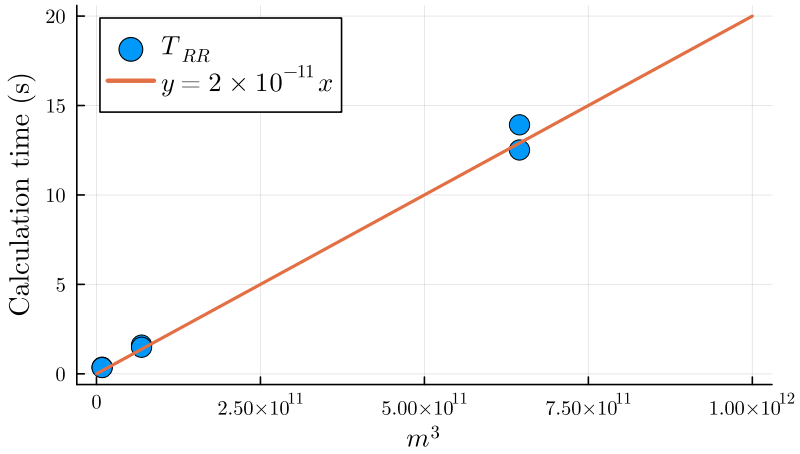


Figure 5.9.: Measured calculating time for one execution of the Rayleigh-Ritz method on GPU calculators with respect to the subspace dimension m.

5.4. Comparison of calculation times

Now that we know the time complexity of the Rayleigh-Ritz step and the Hamiltonian application, we can write an approximation of the time needed for one Chebyshev iteration and one Spectrum Slicing iteration:

$$T_{\text{ChebFi}} = \left(ANN_p + CN_p^2\right) k_c + Bm^3 \tag{5.16}$$

and

$$T_{\text{Slicing}} = \left(ANN_p + CN_p^2\right) k_s + B\left(\frac{m}{n_s}\right)^3. \tag{5.17}$$

For a fixed τ, the degree k_s used for slicing polynomial is given by $k_s = N/m\, \alpha_\tau^{m/N} n_s$. Thus, we can compute the optimal number of slices,

$$n_s^{\text{opt}} = \left(\frac{3B}{\left(ANN_p + CN_p^2\right) N \alpha_\tau^{m/N}}\right)^{1/4} m \tag{5.18}$$

that minimizes T_{Slicing}. This optimum only accounts for one iteration and does not consider the convergence rate μ. However, since in DFT, the convergence rate is mostly governed by the variations of the Hamiltonian operator, this optimum probably makes sense. This n_s^{opt} also fixes the degree $k_s^{\text{opt}} = N/m\, \alpha_\tau^{m/N} n_s^{\text{opt}}$.

5.4.1. Computation time with respect to the number of atoms in the system

In order to reduce the number of parameters, we will express N, N_p and m as a function of the number N_{at} of atoms in the system, which will be our main scaling parameter.

N is the dimension of the matrix H that we want to diagonalize. In DFT, the dimension of H is the number of plane waves in the basis B_G, which is defined as

$$B_G = \left\{ G \in \mathbb{R}^3 \text{ s.t. } G = n_1 b_1 + n_2 b_2 + n_3 b_3 \text{ with } n_1, n_2, n_3 \in \mathbb{Z} \text{ and } \|G\|^2 \leqslant \frac{2E_{\text{cut}}}{4\pi^2} \right\} \tag{5.19}$$

where (b_1, b_2, b_3) are the reciprocal lattice vectors defined by $\langle a_i, b_j \rangle = \delta_{i,j}$ for $i, j = 1, 2, 3$ with a_1, a_2, a_3 the Bravais lattice vectors, that define the unit cell in real space. In Abinit, a_1, a_2 and a_3 are some parameters chosen by the user, depending on the geometry of the system studied. In the case where a_1, a_2 and a_3 form an orthogonal set, we simply have

$$b_1 = \frac{1}{\|a_1\|^2} a_1, \; b_2 = \frac{1}{\|a_2\|^2} a_2 \text{ and } b_3 = \frac{1}{\|a_3\|^2} a_3.$$

The number of basis vectors N can be approximated by the volume of the ellipsoid

$$\left\{ (x, y, z) \in \mathbb{R}^3 \text{ s.t. } \|x b_1 + y b_2 + z b_3\|^2 \leqslant \frac{2E_{\text{cut}}}{4\pi^2} \right\}$$

which is the same as

$$\left\{ (x, y, z) \in \mathbb{R}^3 \text{ s.t. } \frac{x^2}{\|a_1\|^2} + \frac{y^2}{\|a_1\|^2} + \frac{z^2}{\|a_1\|^2} \leqslant \frac{2E_{\text{cut}}}{4\pi^2} \right\}$$

if a_1, a_2 and a_3 form an orthogonal set. This gives

$$N \approx \frac{4}{3}\pi \left(\frac{\sqrt{2E_{\text{cut}}}}{2\pi} \right)^3 \|a_1\| \|a_2\| \|a_3\|.$$

where $\|a_1\| \|a_2\| \|a_3\|$ is the volume of the unit cell, which we write with $V_{\text{at}} N_{\text{at}}$ where V_{at} is the volume of one atom, such that

$$N = \frac{4}{3}\pi \left(\frac{\sqrt{2E_{\text{cut}}}}{2\pi} \right)^3 V_{\text{at}} N_{\text{at}} = c_N N_{\text{at}} \tag{5.20}$$

with

$$c_N = \frac{4}{3}\pi \left(\frac{\sqrt{2E_{\text{cut}}}}{2\pi} \right)^3 V_{\text{at}}$$

a constant that depend on the physical system, it can vary very widely, from a few dozen to a few thousand. With $V_{\text{at}} = 100$ Bohr^3 and $E_{\text{cut}} = 15$ Ha, we have $c_N = 277$.

N_p is the number of projectors in the system which is equal to

$$N_p = n_{\text{lmn}} N_{\text{at}} \tag{5.21}$$

where n_{lmn} is the number of core electrons per atoms (depending one the type of atoms). Each core electron is modeled with one projector. Usually, n_{lmn} varies between 1 and 40.

At zero temperature, the number of sought eigenpairs m is the number of occupied bands and also depends on the number of atoms. When using pseudopotentials, m is the total number of valence electrons so we can write

$$m = n_v N_{\text{at}} \tag{5.22}$$

with n_v the number of valence electron per atoms, that depends on the atom considered and is usually smaller than n_{lmn}. For Titanium, $n_{\text{lmn}} = 18$ and $n_v = 4$.

In DFT calculations, we sometimes need to compute more eigenpairs, for example when the temperature is not 0, or in some other specific applications where "excited" eigenstates are needed for some post-processing. Therefore, we will keep the proportion of sought eigenpairs m/N as a parameter.

Now we can express the total time needed for one step of Spectrum Slicing or one step of Chebyshev filtering with respect to the number of atoms N_{at}. Here, we keep m/N as a parameter. This gives

$$\begin{aligned} T_{\text{ChebFi}} &= \left(AN N_p + C N_p^2\right) k_c + B m^3 \\ &= {N_{\text{at}}}^2 \left(A c_N n_{\text{lmn}} + C {n_{\text{lmn}}}^2\right) k_c + {N_{\text{at}}}^3 B \left(c_N \tfrac{m}{N}\right)^3 \end{aligned} \tag{5.23}$$

and

$$\begin{aligned} T_{\text{Slicing}} &= \left(AN N_p + C N_p^2\right) k_s + B \left(\tfrac{m}{n_s}\right)^3 \\ &= {N_{\text{at}}}^2 \left(A c_N n_{\text{lmn}} + C {n_{\text{lmn}}}^2\right) k_s + {N_{\text{at}}}^3 B \left(c_N \tfrac{m}{N n_s}\right)^3. \end{aligned} \tag{5.24}$$

Now if we focus on Spectrum Slicing with its optimal number of slices, we can express n_s^{opt} and k_s^{opt} in terms of N_{at}, this gives

$$n_s^{\text{opt}} = {N_{\text{at}}}^{1/4} \left(\frac{3B{c_N}^3}{Ac_N n_{\text{lmn}} + C{n_{\text{lmn}}}^2} \right)^{1/4} \frac{m}{N}$$

and

$$k_s^{\text{opt}} = {N_{\text{at}}}^{1/4} \alpha_\tau^{m/N} \left(\frac{3B}{Ac_N n_{\text{lmn}} + C{n_{\text{lmn}}}^2} \right)^{1/4}.$$

Then

$$T_{\text{Slicing}}^{\text{opt}} = {N_{\text{at}}}^{(2+1/4)} \left(Ac_N n_{\text{lmn}} + C{n_{\text{lmn}}}^2 \right)^{3/4} B^{1/4} \left(3^{1/4} \alpha_\tau^{m/N} + {c_N}^{2+1/4} \frac{1}{3^{3/4}} \right). \quad (5.25)$$

Thus $T_{\text{Slicing}}^{\text{opt}}$ scales as ${N_{\text{at}}}^{(2+1/4)}$ while T_{ChebFi} scales as ${N_{\text{at}}}^3$, thus there is always a number of atoms at which Spectrum Slicing becomes more efficient than Chebyshev filtering, for a well chosen number of slices.

Figure 5.10 shows T_{ChebFi} and $T_{\text{Slicing}}^{\text{opt}}$ with respect to the number of atoms, for different parameters. We have displayed two curves for Chebyshev filtering, one with a fix degree $k_c = 6$ and the other with a degree $k_c = \sqrt{k_s^{\text{opt}} \beta_\tau^{m/N} (1 - m/N)}$, that gives the same convergence rate as the optimal Spectrum Slicing. In the domain where Spectrum Slicing becomes profitable, the computation time for Chebyshev filtering is mainly due to the Rayleigh-Ritz procedure, so the degree chosen is not so important. While some E_{cut}, V_{at}, n_{lmn} values are better suited to Spectrum Slicing than others, it is the proportion of sought eigenvalues m/N that makes the biggest difference. At zero temperature, m/N is usually closer to 0.01. This is unfavorable for Spectrum Slicing, that will only become better than Chebyshev filtering for a large, but realistic, number of atoms. However for larger proportion of sought eigenpairs, Spectrum Slicing is readily better the Chebyshev filtering.

Figure 5.11 also illustrate the importance of searching for a high proportion of eigenpairs, by showing the minimal number of atoms to have Spectrum Slicing more efficient that Chebyshev filtering with respect to m/N. The reason for that is that a small m/N most likely yield to thin slices that are more difficult to filter.

5.4.2. Comparison on physical systems

In this section, we will review some particular physical systems of interest, to see if the Spectrum Slicing method can yield better computation times than Chebyshev filtering in this specific cases.

Table 5.3 gives the size parameters of the studied systems. UO2 is a Crystal of uranium

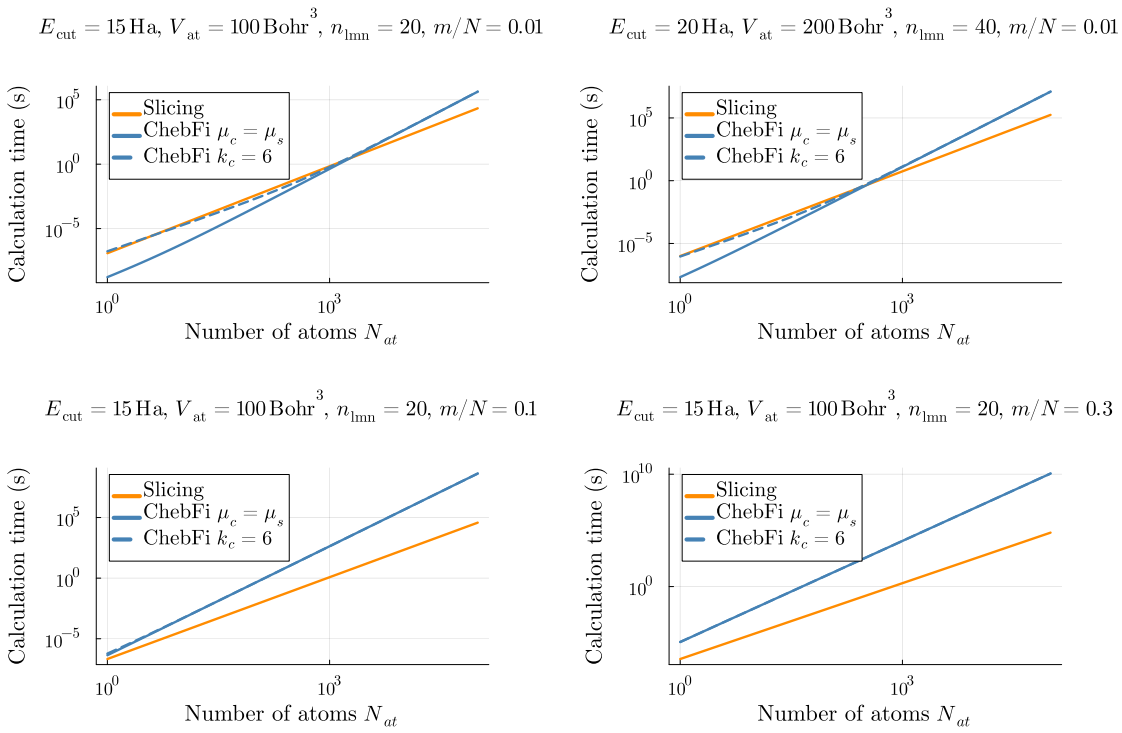


Figure 5.10.: Estimated calculation time with respect to the number of atoms for Spectrum Slicing, Chebyshev filtering with the same convergence rate as Spectrum Slicing and Chebyshev filtering with a degree 6.

dioxide with 54 atoms, it is relatively small and easy to compute with Chebyshev filtering. Au107 is gold in the face-centered cubic crystal structure with 107 atoms ; it is also relatively small. Ti255-A and Ti255-B are titanium crystals (in hexagonal close-packed structure). In case B, E_{cut} is increased and we compute more eigenstates (m), which will lead to more precise results at high temperature. This two cases are already expensive to compute. Finally, Ga2O3-A and Ga2O3-B are gallium (III) oxides. In case B, the number of atoms in the unit cell is increased, which drastically increases the number of plane waves N needed to discretize the Hamiltonian. The number m of eigenpairs to compute which scales with the number of atoms is also increased significantly. This two cases are very expensive to run and Ga203-B could not yet be computed.

Tables 5.4 and 5.5 show the estimated computation times of one diagonalization step with Chebyshev filtering and Spectrum Slicing, for different values of k_c, the degree of the polynomial in the Chebyshev filtering method, and the corresponding slicing parameters (chosen to match the convergence rate given by k_c with Chebyshev filters). The empty cells in the table correspond to cases where it is impossible to filter the part of the spectrum we

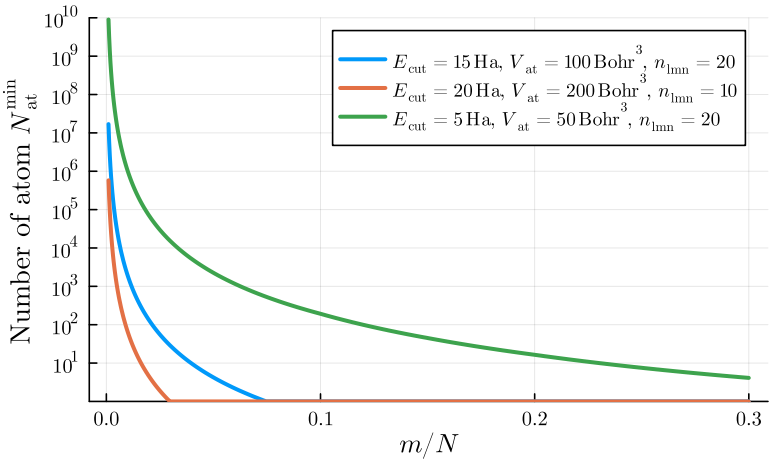


Figure 5.11.: Estimated minimal number of atoms to have the optimal Spectrum Slicing become more profitable than Chebyshev filtering, with respect to the proportion of sought eigenpairs m/N, for different size parameters E_{cut}, V_{at} and n_{lmn}.

Physical systems	N_{at}	N_p	n_{lmn}	m	N	m/N	E_{cut}	V_{at}
UO2	54	1224	22,7	256	13543	0,0189	15	90,3
Au107	107	1926	18	640	33804	0,0189	15	114
Ti255-A	255	4590	18	2048	17092	0,120	5	126
Ti255-B	255	4590	18	4096	34590	0,118	8	126
Ga2O3-A	1960	23520	12	8640	77396	0,112	5	73,9
Ga2O3-B	4160	49920	12	18432	164270	0,112	5	74

Table 5.3.: Size parameters of some physical systems.

are looking for using polynomials of degree $k_s = {k_c}^2 \left(\beta_\tau^{m/N}(1 - m/N)\right)^{-1}$ (for $\tau = 0,5$).

We observe that for the two smallest systems UO2 and Au107, the Spectrum Slicing method is never cost-effective compared with the Chebyshev filtering method. Indeed, since m is quite small, the computation time is dominated by the filtering step and not by the Rayleigh-Ritz step (see table 5.6). For the other systems, the Spectrum Slicing method becomes profitable for $k_c = 8$ and $k_c = 16$ i.e. for a number of slices between 5 and 20 approximately. When the number of slices is increased to around 80, the time saved in the Rayleigh-Ritz step no longer compensate for the very high degree filter. This shows that the number of slices must be chosen wisely. We see that the bigger the system is, the bigger is the time difference between the two methods. For Ga2O3-B, with $k_c = 8$, we can expect

the computation time to be reduced by a factor 14, which would make its execution time similar to Ga2O3-A with Chebyshev filtering, thus making it possible to run.

	$k_c = 4$				$k_c = 8$			
Physical systems	T_{ChebFi} (s)	T_{Slicing} (s)	k_s	n_s	T_{ChebFi} (s)	T_{Slicing} (s)	k_s	n_s
UO2	0,000714	-	-	-	0,00109	-	-	-
Au107	0,00647	-	-	-	0,00770	-	-	-
Ti255-A	0,175	0,19	21	1	0,179	0,0742	84	5
Ti255-B	1,38	1,40	21	1	1,38	0,103	83	5
Ga2O3-A	12,0	13,4	21	1	13,0	1,89	81	5
Ga2O3-B	126	127	21	1	126	9,05	81	5

Table 5.4.: Estimated computation time for one eigensolver step, for a degree $k_c = 4$ and $k_c = 8$ in the Chebyshev method and the corresponding parameters in the slicing method ($\tau = 0,5$).

	$k_c = 16$				$k_c = 32$			
Physical systems	T_{ChebFi} (s)	T_{Slicing} (s)	k_s	n_s	T_{ChebFi} (s)	T_{Slicing} (s)	k_s	n_s
UO2	0,00185	0,0164	173	3	0,00337	0,0654	691	14
Au107	0,0101	0,0532	173	3	0,0150	0,212	691	14
Ti255-A	0,186	0,289	333	21	0,200	1,16	1332	87
Ti255-B	1,39	0,367	331	21	1,41	1,46	1323	86
Ga2O3-A	13,3	7,08	321	20	13,6	28,3	1282	80
Ga2O3-B	127	32,0	322	20	128	128	1286	81

Table 5.5.: Estimated computation time for one eigensolver step, for a degree $k_c = 16$ and $k_c = 32$ in the Chebyshev method and the corresponding parameters in the slicing method ($\tau = 0,5$).

All of this time estimations have been made considering a perfect (and infinite) parallelization, with every slice computed in parallel and each application of the filters to the vectors done in parallel on all the vectors, in each slice. If the parallelization is not perfect, the time spent on filtering is multiplied, penalising the Spectrum Slicing method more than the Chebyshev method. Table 5.6 shows the details of the computation time on both methods. We see that for example in the Ga2O3-B case, if each processor has more than

15 eigenvectors to filter (thus filtering them sequentially), Spectrum Slicing is no longer profitable. However the number of process needed to make the computations with full parallelization is achievable with GPUs.

	Chebyshev filtering			Slicing methode		
Physical systems	T_{tot} (s)	$T_{\text{filtering}}$ (s)	T_{RR} (s)	T_{tot} (s)	$T_{\text{filtering}}$ (s)	T_{RR} (s)
UO2	0,179	0,000757	0,000336	-	-	-
Au107	1,38	0,00245	0,00524	-	-	-
Ti255-A	0,179	0,00694	0,172	0,0742	0,0729	0,00137
Ti255-B	1,38	0,00887	1,374	0,103	0,0920	0,011
Ga2O3-A	13,0	0,176	12,9	1,89	1,79	0,1031
Ga2O3-B	126	0,795	125	9,05	8,05	1,00

Table 5.6.: Estimated computation time for a diagonalization step, for a degree $k_c = 8$ in the Chebyshev method and the corresponding parameters in the slicing method ($\tau = 0,5$) with details of the time spent on filtering and Rayleigh-Ritz.

In this section we have shown the Spectrum Slicing method can be more time effective than Chebyshev filtering in some specific cases, namely for very large systems or smaller systems with a large proportion of sought eigenpairs. Spectrum Slicing is well suited, for instance, for high-temperature calculations.

However, all these estimates have been made with a number of simplifications, the most important of which is likely that of homogeneously distributed eigenvalues. Also, in actual computation, we would compute more eigenpairs than needed in Chebyshev filtering and in each slice when using the Spectrum Slicing method, to improve convergence. Thus the next step would be to compare these estimates with reality by implementing the Spectrum Slicing method in Abinit. This will allow us to test the relevance of our approximations. In addition, we could then also observe the effects of communications between processors, which have not been considered at all here.

6. Conclusion

This Master's thesis reviewed the *Spectrum Slicing* method and its application for DFT calculations in plane waves. The study of this specific eigensolver was motivated by the observation of the calculation time allocated to the Rayleigh-Ritz step in DFT calculations, in particular the evolution of this time during the transition from purely CPU calculators to CPU+GPU calculators. We have found that, while the other computational steps are reduced by the addition of GPUs in the two iterative solvers implemented in Abinit, namely LOBPCG and ChebFi, this is not the case for the Rayleigh-Ritz step. So, if the computation time of this step was not to be taken into account until now, this is no longer the case. We need an iterative solver which reduces and/or parallelizes the calculation time related to the Rayleigh-Ritz. This is the whole point of the Spectrum Slicing method.

In this study, we were particularly interested in applying this method to dense matrices. This adds complexity because it becomes very difficult to invert the matrices and even to store them. Moreover, using this solver within the DFT framework adds certain stability and optimization challenges, as the matrix to be diagonalized is no longer constant.

The first step in this work was a bibliographical study of iterative eigensolvers, in particular "Subspace iteration" type of methods, of which Spectrum Slicing belongs. This allowed me to take the subject in hand. A large part of this work is therefore a study and description of the variety of existing strategies for Slicing methods. The first sections of this work report on this bibliographical study.

A second important part of the work was the implementation of a prototype in Julia to test and experiment with the method. This highlighted a number of difficulties, in particular the importance of choosing the right slices and correctly merging the results obtained on each processor, especially in DFT calculations. We were also able to verify that plane-wave DFT calculations converge with the Spectrum Slicing eigenvalue solver.

Finally, the last but not least aim of this work was to assess the usefulness and applicability of this method. This was done in the last section. Indeed, while Spectrum Slicing reduces the computational load of Rayleigh-Ritz, the other steps are extended. With our naive modeling, we were able to estimate that Spectrum Slicing will be more efficient than Chebyshev filtering (considered here as a reference) in certain cases. For Spectrum Slicing

to be of interest, the problem must be of large size, so that the Rayleigh-Ritz computation time has a significant weight in the Chebyshev method, and the proportion of eigenstates sought must be large enough not to penalize the filtering too much. This corresponds to specific applications, like high temperatures calculation for instance.

The implementation of this iterative eigensolver in Abinit was not part of this work. It should be the next step as it will allow us to test Spectrum Slicing on larger systems and also to compare our complexity model with reality. We have made a number of simplifications to estimate computation times with Chebyshev Filtering and Spectrum Slicing. In particular, we have assumed an homogeneous distribution of eigenvalues, which is never the case for real materials and the distribution has a large influence on the choice of slices and therefore on the convergence rate and on filtering complexity. Regarding the time complexity of the paralellization, we have also not taken into account the communications between processors. These increase computation time.

In most of this work, we were interested in solving standard eigenproblems. However, in PAW formalism for instance, we need to solve generalized eigenvalue problems. These can always be rewritten as standard problems by inverting the overlap matrix, but some filters (e.g. rational filters) can be directly adapted to generalized eigenvalue problems. The effect of adding an overlap matrix needs to be studied since the PAW formalism is widely used in Abinit.

A. Chebyshev polynomials

In this appendix, we will study Chebyshev polynomials, that are use full to build the eigensolvers used in this work.

A.1. Chebyshev polynomials of the first and second kind

Definition A.1.1. *Chebyshev polynomials of the first kind are defined by the following recurrence relation*

$$\begin{aligned} &T_0(X) = 1 \\ &T_1(X) = X \\ &\forall n \geqslant 1,\, T_{n+1}(X) = 2XT_n(X) - T_{n-1}(X). \end{aligned} \tag{A.1}$$

Lemma A.1.1. *For θ in $\mathbb{R}$, Chebyshev polynomials of the first kind are such that, for $n \geqslant 0$,*

$$\cos(n\theta) = T_n(\cos\theta) \tag{A.2}$$

and

$$\cosh(n\theta) = T_n(\cosh\theta). \tag{A.3}$$

Proof. This is proven by induction. Let $\theta \in \mathbb{R}$, we have $\cos 0 = 1 = T_0(\cos\theta)$, $\cos\theta = T_1(\cos\theta)$, $\cosh 0 = 1 = T_0(\cosh\theta)$ and $\cosh\theta = T_1(\cosh\theta)$. Then we assume that $\cos n\theta = T_n(\cos\theta)$ and $\cos(n-2)\theta = T_{n-2}(\cos\theta)$, then, then using the usual trigonometry formulae,

$$\begin{aligned} \cos(n+1)\theta &= \cos n\theta\cos\theta - \sin n\theta\sin\theta \\ &= \cos n\theta\cos\theta - [\cos((n-1)\theta)\sin\theta + \sin((n-1)\theta]\cos\theta)\sin\theta \\ &= \cos n\theta - \cos((n-1)\theta)(1-\cos^2\theta) + \sin((n-1)\theta)\sin\theta\cos\theta \\ &= \cos n\theta\cos\theta + \cos\theta[\sin((n-1)\theta)\sin\theta - \cos((n-1)\theta)\cos\theta)] - \cos((n-1)\theta) \\ &= 2\cos n\theta\cos\theta - \cos((n-1)\theta) \\ &= T_n(\cos\theta)\cos\theta - T_{n-1}(\cos\theta) \\ &= T_{n+1}(\cos\theta) \end{aligned} \tag{A.4}$$

and similarly, we assume that $\cosh n\theta = T_n(\cosh\theta)$ and $\cosh(n-2)\theta = T_{n-2}(\cosh\theta)$, then using the usual hyperbolic trigonometry formulae,

$$\begin{aligned}\cosh(n+1)\theta &= \cosh n\theta\cosh\theta + \sinh n\theta\sinh\theta \\ &= \cosh n\theta\cosh\theta + [\sinh((n-1)\theta)\cosh\theta + \sinh\theta\cosh((n-1)\theta)]\sinh\theta \\ &= \cosh n\theta + \cosh\theta[\sinh((n-1\theta)\sinh\theta + \cosh\theta\cosh((n-1)\theta)] - \cosh((n-1)\theta) \\ &= 2\cosh(n\theta)\cosh(\theta) - \cosh((n-1)\theta) \\ &= T_{n+1}(\cosh\theta)\end{aligned} \tag{A.5}$$

which concludes the proof. □

Definition A.1.2. *Chebyshev polynomials of the second kind are defined by the following recurrence relation*

$$\begin{aligned}&U_0(X) = 1 \\ &U_1(X) = 2X \\ &\forall n \geqslant 1,\, U_{n+1}(X) = 2XU_n(X) - U_{n-1}(X).\end{aligned} \tag{A.6}$$

Lemma A.1.2. *For θ in $]0, 2\pi[$, Chebyshev polynomials of the second kind are such that, for $n \geqslant 0$,*

$$U_n(\cos\theta) = \frac{\sin(n+1)\theta}{\sin\theta}. \tag{A.7}$$

Proof. This is proven by induction. Let θ in $\mathbb{R}$, we have

$$U_0(\cos\theta) = 1 = \frac{\sin\theta}{\sin\theta} \tag{A.8}$$

and

$$U_1(\cos\theta) = 2\cos\theta = 2\frac{\cos\theta\sin\theta}{\sin\theta} = \frac{\sin 2\theta}{\sin\theta}. \tag{A.9}$$

Then for n in $\mathbb{N}$ we have

$$\begin{aligned}
\sin(n+1)\theta &= \cos n\theta \sin\theta + \cos\theta \sin n\theta \\
&= \sin\theta\cos\theta\cos(n-1)\theta + \sin\theta\sin\theta\sin(n-1)\theta + \cos\theta\sin n\theta \\
&= \sin\theta\cos\theta\cos(n-1)\theta + (1-\cos^2\theta)\sin(n-1)\theta + \cos\theta\sin n\theta \\
&= \cos\theta\left(\sin\theta\cos(n-1)\theta + \cos\theta\sin(n-1)\theta + \sin n\theta\right) - \sin(n-1)\theta \qquad \text{(A.10)} \\
&= 2\cos\theta\sin n\theta - \sin(n-1)\theta \\
&= \sin\theta\left(2\cos\theta U_n(\cos\theta) - U_{n-1}(\cos\theta)\right) \\
&= \sin\theta U_{n+1}(\cos\theta).
\end{aligned}$$

Hence the induction is proven. □

Lemma A.1.3. *For $n \geqslant 1$ and $t \in \mathbb{R}$, $T_n'(t) = nU_{n-1}(t)$.*

Proof. Let $\theta \in \mathbb{R}$ and $n \geqslant 0$, $T_n(\cos\theta) = \cos(n\theta)$ so

$$\frac{d}{d\theta}\cos n\theta = -n\sin n\theta = -T_n'(\cos\theta)\sin\theta \qquad \text{(A.11)}$$

thus,

$$T_n'(\cos\theta) = \frac{n\sin n\theta}{\sin\theta} = U_{n-1}(\cos\theta). \qquad \text{(A.12)}$$

The equality is proved on $[-1,1]$ and by d'Alembert–Gauss theorem, the equality is proved on $\mathbb{R}$. □

Lemma A.1.4. *For $n \geqslant 0$,*

$$U_n(1) = n+1 \qquad \text{(A.13)}$$

$$U_n(-1) = (-1)^n(n+1) \qquad \text{(A.14)}$$

Proof. We have $U_0(1) = 1$, $U_1(-1) = 2$ and for $n \geqslant 0$

$$U_{n+2}(1) = 2U_{n+1}(1) - U_n(1) = 2(n+2) - (n+1) = n+3.$$

Then in -1 we have $U_0(-1) = 1$, $U_1(-1) = -2$ and for $n \geqslant 0$

$$U_{n+2}(1) = 2U_{n+1}(1) - U_n(1) = 2(-1)^{n+1}(n+2) - (-1)^n(n+1) = (-1)^{n+2}(n+3).$$

□

A.2. Chebyshev series expansion

To study Chebyshev series expansion, we mostly used the book [Riv20].

A.2.1. Orthogonality of Chebyshev Polynomials

The weight function associated with Chebyshev polynomials of the first kind is

$$w : \begin{array}{ccc} [-1,1] & \to & \mathbb{R}^+ \\ t & \mapsto & \frac{1}{\sqrt{1-t^2}} \end{array} . \tag{A.15}$$

Using this weight function, we define the following scalar product.

Definition A.2.1 (Chebyshev scalar product)**.** *For f, g real functions defined on $[-1,1]$,*

$$\langle f, g\rangle_w = \int_{-1}^{1} f(t)g(t)w(t)dt \tag{A.16}$$

Theorem A.2.1. *Chebyshev polynomial of the first kind form an orthogonal system with respect to the Chebyshev scalar product $\langle\cdot,\cdot,\rangle_w$.*

Proof. Let $m, n \in \mathbb{N}$,

$$\langle T_n, T_m\rangle = \int_{-1}^{1} T_n(x)T_m(x)\frac{1}{\sqrt{1-x^2}}dx = \int_0^{\pi} \cos(n\theta)\cos(m\theta)d\theta \tag{A.17}$$

with the substitution $x = \cos\theta$. Then using $\cos a \cos b = \frac{1}{2}(\cos(a+b)+\cos(a-b))$, we have

$$\langle T_n, T_m\rangle = \frac{1}{2}\int_0^{\pi} \cos((n+m)\theta) + \cos((n-m)\theta)d\theta \tag{A.18}$$

If $n \neq m$ this gives

$$\langle T_n, T_m\rangle = \left[\frac{\sin((n+m)\theta)}{n+m}\right]_0^{\pi} + \left[\frac{\sin((n-m)\theta)}{n-m}\right]_0^{\pi} = 0 \tag{A.19}$$

otherwise if $n = m \geqslant 1$,

$$\langle T_n, T_n\rangle = \frac{1}{2}\int_0^{\pi} 1d\theta = \frac{\pi}{2} \tag{A.20}$$

and

$$\langle T_0, T_0\rangle = \frac{1}{2}\int_0^{\pi} 2d\theta = \pi \tag{A.21}$$

□

Furthermore, since Chebyshev polynomials form a basis of the vector space $\mathbb{R}[X]$, we have from the Weierstrass approximation theorem, that Chebyshev polynomials form a complete orthogonal system on $[-1, 1]$.

A.2.2. Formal Chebyshev series expansion

Since Chebyshev polynomials form a complete orthogonal system on $[-1, 1]$, we can define Chebyshev polynomial series expansion as generalized Fourier series. Let f a function defined on $[-1, 1]$, its Chebyshev series expansion is defined as the formal sum

$$\tilde{f} = \sum_{i=0}^{\infty} c_i(f) T_i \tag{A.22}$$

where

$$c_i(f) = \frac{\langle f, T_i \rangle_w}{\langle T_i, T_i \rangle_w}. \tag{A.23}$$

A.2.3. Relation between Chebyshev series expansion and Fourier series expansion

Let $f : [-1, 1] \to \mathbb{R}$ a function, We define

$$\begin{array}{rccc} & \mathbb{R} & \to & \mathbb{R} \\ g : & & & \\ & \theta & \mapsto & f(\cos\theta). \end{array} \tag{A.24}$$

Thus, g is a function with period 2π and its real Fourier series expansion is

$$\frac{1}{2}a_0 + \sum_{i=1}^{\infty} a_i \cos i\theta + b_i \sin i\theta \tag{A.25}$$

with, for $i \geqslant 0$,

$$a_i = \frac{1}{\pi} \int_{-\pi}^{\pi} g(\theta) \cos i\theta d\theta \tag{A.26}$$

and

$$b_i = \frac{1}{\pi} \int_{-1}^{1} g(\theta) \sin i\theta d\theta = 0 \tag{A.27}$$

since g is even. So the Fourier series reduces to the cosine expansion

$$\frac{1}{2}a_0 + \sum_{i=1}^{\infty} a_i \cos i\theta. \tag{A.28}$$

Then, using the substitution $\theta = \arccos t$ and the fact that g is even, we have

$$a_i = 2 \times \frac{1}{\pi} \int_{-1}^{1} g(\arccos t) \cos(i \arccos t) \frac{dt}{\sqrt{1-t^2}} = \frac{2}{\pi} \int_{-1}^{1} f(t) T_i(t) \frac{dt}{\sqrt{1-t^2}}. \tag{A.29}$$

This means that the Chebyshev series expansion of f is the same as the Fourier cosine series expansion of g. Thus, we will be able to apply known convergence properties of Fourier series to Chebyshev series.

A.2.4. Convergence of Chebyshev series expansion

Theorem A.2.2 (Jordan-Dirichlet)**.** *If f is a periodic function of bounded variation on each period, its Fourier series converges at each point t to*

$$\lim_{\varepsilon \to 0} \frac{f(t+\varepsilon) + f(t-\varepsilon)}{2}. \tag{A.30}$$

In particular, this means that we have convergence for all piecewise $\mathcal{C}^1$ functions, as for instance $\chi_{[a,b]}$ for two real $a < b$.

A.2.5. Damping

Expansion of discontinuous functions in Fourier series lead to oscillations near the discontinuity. This is known as the Gibbs phenomenon. These unwanted oscillations are present in Chebyshev series expansions and can be attenuated using well chosen damping coefficients, that is by replacing

$$\sum_{i=0}^{k} c_i T_i \tag{A.31}$$

with

$$\sum_{i=0}^{k} d_i^k c_i T_i \tag{A.32}$$

for some smoothing coefficients d_i^k. Here, we will present two choices of damping coefficient that are Jackson damping and σ-damping.

Jackson damping Jackson damping is presented in [Jac30] and [Riv81].

σ-damping σ-damping is presented in [Lan88] and defined as follows :

$$d_0^k = \sigma_0^k = 1 \tag{A.33}$$

and for $j = 1, \ldots, k$,

$$d_i^k = \sigma_i^k = \frac{\sin i\theta_k}{i\theta_k} = \operatorname{sinc} i\theta_k \text{ with } \theta_k = \frac{\pi}{k+1}. \tag{A.34}$$

This will attenuate the higher frequency coefficient, thus reducing the oscillations. More precisely, the sine cardinal function sinc is the Fourier transform of a characteristic function which means that that the sum $\sum_{i=0}^{k} c_i \sigma_i^k \cos i\theta$ is the truncated Fourier series of g convoluted with the function $\frac{\pi}{2\theta_k}\chi_{[-\theta_k,\theta_k]}$, of integral π, which can be seen as smoothing of g.

B. Code

This appendix contains some of the code functions written in *Julia* language used to test and study the Spectrum Slicing method. The code is not given in its entirety. However, it is a good illustration of the points covered in this work. For the DFT, we used the DFTK package [HLC21], which allows you to set a custom eigensolver as parameter.

Slices

```
function maxDOSClusteringSlices(lambda, ns, lmin, lmax; n=1000)
    # Returns at most ns slices, according to lambda
    nv = length(lambda)
    phi = DOS(lambda)
    blast = min(maximum(lambda), lmax) # It doesn't matter if the max is
    underestimated
    T = range(lmin, blast; length=n)
    phi_T = [phi(t) for t in T]
    mins = findLocalMinima(phi_T)
    mins = [1; mins]     # adding lmin index
    push!(mins, n)       # adding blast index
    l = length(mins) - 1
    ms = [ integral(mins[i], mins[i+1], T, phi_T)*nv for i in 1:l]
    S = []
    for i in 1:l
        push!(S, (T[mins[i]], T[mins[i+1]], ms[i]))
    end
    while l > ns
        ms_merged = [ms[i] + ms[i+1] for i in 1:l-1]
        i_min = argmin(ms_merged)
        s_new = (S[i_min][1], S[i_min+1][2], ms_merged[i_min])
        S[i_min] = s_new
        ms[i_min] = ms_merged[i_min]
        deleteat!(S, i_min+1)
        deleteat!(ms, i_min+1)
        l -= 1
    end

    return S
```

```
end

function slicesUpdate(H, X0, ns, lmin, lmax, X0s_old;
        return_info=false,
        slicesDist=maxDOSClusteringSlices,
        n_extra=1,
        random=0,
        args...
    )
    # Slices chosen depending on the previous slices with slicesDist

    lambda = RayleighQuotients(H, X0) # Liste of Rayleigh quotients

    S = slicesDist(lambda, ns, lmin, lmax; args...)
    blast = last(S)[2]
    ns = length(S)

    # New X0s allocation
    X0s_old_concat = X0s_old[1]
    for i in 2:ns
        X0s_old_concat = [X0s_old_concat X0s_old[i]]
    end
    lambdas_old_concat = RayleighQuotients(H, X0s_old_concat)
    I_sup = findall(lambda -> lambda .> blast, lambdas_old_concat) #
    eigenvalues indexes (lambdas_old_concat) higher than the new slices
    X = X0
    # We add to X and lambda the eigenvectors/values calculated in previous
    iterations greater than blast
    # These vectors correspond to lambda_m+1, lambda_m+2 ... we don't need
    to calculate them, but they are useful for convergence.
    if !isnothing(I_sup)
        ress_old_concat = Residuals(H, lambdas_old_concat, X0s_old_concat)
        res = Residuals(H, lambda, X0)
        lambda_sup, X_sup, = removeDuplicate(blast, lambda, X0, res,
    lambdas_old_concat[I_sup], X0s_old_concat[:, I_sup], ress_old_concat[
    I_sup], []; right=true)
        X = [X0 X_sup]
        lambda = vcat(lambda, lambda_sup)
    end

    p = sortperm(lambda)
    X = X[:, p]
    lambda = lambda[p]

    n = size(X, 2)
```

```
    N = size(X, 1)
    # Construction of new subspaces X0s
    X0s_new = []
    for i in 1:ns
        (a, b, m) = S[i]
        i1 = findfirst(lambda -> lambda .>= a, lambda)
        i2 = findlast(lambda -> lambda .<= b, lambda)
        X0i = X[:, i1:i2]

        # We add (n_extra) vectors to make the slice "overflow".

        # New random vectors
        Xnew_rand = rand(ComplexF64, (N, n_extra))
        # Already calculated vectors that are the closest to the slice
        d = [ min(abs(a-lambda[i]), abs(lambda[i]-b)) for i in 1:n]
        dist_slice = [ ( i1 <= i <= i2 ? Inf : d[i] ) for i in 1:n]
        p = partialsortperm(dist_slice, 1:n_extra)
        Xnew_old = X[:, p]
        # Mixing the two
        Xnew = random * Xnew_rand + (1-random) * Xnew_old

        X0i = [X0i Xnew]

        push!(X0s_new, X0i)
    end

    if return_info
        return S, X0s_new, "Slices update"
    else
        return S, X0s_new
    end
end
```

Filters

```
    function scalarChebyDirac(t, k, gamma; damping=sigma, arg...)
  # entre -1 et 1
  T = Chebyshev(t, k)
  Tg = Chebyshev(gamma, k)
  rho = 1/2*damping(0,k)*T[1]
  rhog = 1/2*damping(0,k)*Tg[1]
  for j in 2:k+1
    mu = cos((j-1)*acos(gamma))
    rho += mu*damping(j-1,k)*T[j]
    rhog += mu*damping(j-1,k)*Tg[j]
```

```
  end
  return rho/rhog
end

function degChebyDirac(a_, b_; tau=0.5, maxDeg=200, damping=sigma)
  gamma = (a_+b_)/2
  Ta = Chebyshev(a_, maxDeg)
  Tb = Chebyshev(b_, maxDeg)
  Tg = Chebyshev(gamma, maxDeg)
  rhoa = 1/2*Ta[1]
  rhob = 1/2*Tb[1]
  rhog = 1/2*Tg[1]
  k=1
  while ( abs(rhoa/rhog) > tau && abs(rhob/rhog) > tau && k < maxDeg)
    k += 1
    mu = cos((k-1)*acos(gamma))
    rhoa += mu*Ta[k]
    rhob += mu*Tb[k]
    rhog += mu*Tg[k]
  end
  k = max(k, 5) # degree at least 5
  f = x -> scalarChebyDirac(b_, k, x; damping)-scalarChebyDirac(a_, k, x;
   damping)
  delta = 0 #(b_-a_)*1e-5 # to avoid calculating f to close to -1
    if f(max(a_, delta-1))*f(min(b_, 1-delta)) < 0
      gamma = find_zero(f, (max(a_, delta-1), min(b_, 1-delta)))
  else
    @warn "Filter not centered : some eigenvalues could be missed"
    end
  return k, gamma
end

function ChebyDirac(H, a, b, lmin, lmax; damping=one, tau=0.8, maxDeg=200,
   deg=0, gam=0, count_matvec=false)
  c = (lmin+lmax)/2
  r = (lmax-lmin)/2
  a_ = (a-c)/r
  b_ = (b-c)/r
  if deg > 0
    if gam == 0
      gam = (a_+b_)/2
    end
  else
    deg, gam = degChebyDirac(a_, b_; tau=tau, maxDeg=maxDeg, damping)
  end
```

```
54
55    function filterH(V; count_matvec=false)
56      THV = [V, 1/r*(H*V-c*V)]
57      FHV = 1/2*damping(0, deg)*THV[1]
58      FHV += gam*damping(1, deg)*THV[2]
59      Tg = Chebyshev(gam, deg)
60      fg = 1/2*g(0,deg)*Tg[1] + gam*damping(1, deg)*Tg[2]
61      for i = 3:deg+1
62        mu = cos((i-1)*acos(gam))
63        THV = push!(THV, (2/r*(H*THV[i-1]-c*THV[i-1])-THV[i-2]))
64        FHV += mu*damping(i-1, deg)*THV[i]
65        fg += mu*damping(i-1, deg)*Tg[i]
66      end
67      if count_matvec
68        return FHV./fg, deg
69      else
70        return FHV./fg
71      end
72    end
73
74    return filterH
75  end
```

Spectrum slicing eigensolver

```
1
2   function subspaceProj(filterH, Q; count_matvec=false)
3     m = size(Q, 2)
4     if count_matvec
5       fQ, n_matvec = filterH(Q; count_matvec)
6       V = qr(fQ).Q[:, 1:m] # Orthonormalisation
7       return V, n_matvec
8     else
9       fQ = filterH(Q; count_matvec)
10      V = qr(fQ).Q[:, 1:m] # Orthonormalisation
11      return V
12    end
13  end
14
15  function eigenSolverSlice(H, X0, a, b, lmin, lmax;
16      filter=ChebyDirac,
17      argf=Dict(),    # Dictionnary containing the keyword arguments of filter
18      maxiter=100,
19      miniter=1,
20      tol=1e-10,
```

```
    to=DFTK.timer
  )
  N, nv = size(X0)
  @debug "  Entering eigenSolverSlice"
  # Filtrage
  filterH = filter(H, a, b, lmin, lmax; argf...)

  # First filtering
  n_matvec = 0
  @timeit to "Filtering" X, n = subspaceProj(filterH, X0; count_matvec=true)
  n_matvec += n
  deg = n
  @timeit to "Rayleigh-Ritz" lambda, X, res, n = RayleighRitz(X, H;
    count_matvec=true)
  n_matvec += n
  res_hist = zeros(maxiter+1, nv)

  max_res = maximum(res)
  n_ite = 1
  @debug "  First iteration done"
  while (max_res > tol && n_ite < maxiter) || n_ite < miniter
    # Projection
    @timeit to "Filtering" X, n = subspaceProj(filterH, X; count_matvec=true
    )
    n_matvec += n
    # Rayleigh-Ritz
    @timeit to "Rayleigh-Ritz" lambda, X, res, n = RayleighRitz(X, H;
    count_matvec=true)
    n_matvec += n
    res_hist[n_ite, :] = res
    n_ite += 1

    max_res = maximum(res)
  end

  (; lambda, X, res, res_hist, n_matvec, iterations=n_ite, converged=(
    max_res<tol), deg)
end

function slicingEigenSolver(H, X0;
    ns=1,
    filter=ChebyDirac,    # Filtre
    argf=Dict(),      # Keyword arguments pour le filtre
    n_extra=0,
    slices=maxDOSClusteringSlices,
```

```
62      random=0,
63      n_iterations=nothing,
64      n_update=16,
65      p=0.1,
66      save_logs=true,
67      exact_eigenvalues=false,
68      to=DFTK.timer,
69      solving_data=nothing, # Must be specified : used to
70                                      #know the current kpoint and SCF step
71                                      # store information about the previous
    iteration that will be needed
72                                      # store the solving history
73      maxiter=100,
74      miniter=1,
75      tol=1e-10,
76      prec=I,
77      n_conv_check=nothing
78  )
79
80  N, nv = size(X0)
81  if !isnothing(n_iterations) # constant number of iterations per SCF step
82      maxiter = miniter = n_iterations
83  end
84
85  n_matvec = 0  # warning : matvec count not done correctly in all the code
86
87  lmin, lmax, n = spectrumBounds(H; count_matvec=true)
88  n_matvec += n
89  i_kpt = solving_data[:n_slicing][] % solving_data[:nkpt] + 1    # Index of
      the current kpoint
90  i_ite = div(solving_data[:n_slicing][], solving_data[:nkpt]) +1 # Index of
      the SCF iteration on this kpoint
91
92  # Slices
93  info_slices = ""
94      if i_ite == 1                           # Slices initialization
95      @timeit to "slicesInit" S, X0s, info_slices, n = slicesInit(H, X0, ns,
    lmin, lmax; slicesDist=slices, return_info=true, n_extra, count_matvec=
    true)
96      n_matvec += n
97  elseif (solving_data[:update_slices])[i_kpt] > n_update   # Slices update
98      X0s_old = last((solving_data[:Xs_history])[i_kpt])
99      @timeit to "slicesUpdate" S, X0s, info_slices = slicesUpdate(H, X0, ns,
    lmin, lmax, X0s_old; random, slicesDist=slices, n_extra, return_info=
    true)
```

```
100     (solving_data[:update_slices])[i_kpt] = 0
101   else                              # Slices fixes
102     S_old = last((solving_data[:S_history])[i_kpt])
103     X0s_old = last((solving_data[:Xs_history])[i_kpt])
104     missing_vectors = (solving_data[:missing_vect])[i_kpt]
105     @timeit to "slicesConst" S, X0s, info_slices = slicesConst(S_old,
        X0s_old, missing_vectors; p, return_info=true)
106     (solving_data[:update_slices])[i_kpt] *= 2
107     lmin = S[1][1]
108     lmax = max(lmax, last(S)[2])
109   end
110
111   ns = length(S)  # We might have less slice than wanted (if there is less
        than ns clusters of eigenvalues)
112   converged = true
113   iterations = 0
114   lambdas = []
115   Xs = []
116   ress = []
117   residual_history = []
118   missing_vect = [ true for j in 1:ns ]
119   degree = []
120   info_remove = ["" for i in 1:ns]
121
122   result_slices = []
123
124   # Solving the eigenvalue problem on each slice
125   n_matvec_max = 0
126   for i in 1:ns
127         (a, b, m) = S[i]
128     X0i = X0s[i]
129
130     @timeit to "eigenSolverSlice" eSlice = eigenSolverSlice(H, X0i, a, b,
        lmin, lmax;
131       filter,
132       argf,
133       maxiter,
134       miniter,
135       tol,
136       to
137     )
138
139     lambda, X, res, res_hist  = eSlice.lambda, eSlice.X, eSlice.res, eSlice.
        res_hist
140
```

```
141     push!(lambdas, copy(lambda))
142     push!(Xs, copy(X))
143     push!(ress, copy(res))
144     push!(degree, eSlice.deg)
145
146     # Removing vectors that are outside the slice
147     if i < ns
148       @timeit to "removeOutside" lambda, X, res, list, info, removed =
      removeOutside2(a, b, lambda, X, res, [res_hist]; return_info=true) #
      removing eigenpairs that are for sure outside the slice
149       res_hist = list[]
150       info_remove[i] *= " "*info
151     else
152       # last slice : we don't remove vectors that are to big because
153       # they will be removed at the end with remove_extra
154       @timeit to "removeOutsideInf" lambda, X, res, list, info, outside =
      removeOutsideInf(a, b, lambda, X, res, [res_hist]; return_info=true) #
      removing eigenpairs that are for sure below the slice
155       res_hist = list[]
156       info_remove[i] *= " "*info
157       removed = outside # Outside vectors exist (maybe not removed)
158     end
159
160     missing_vect[i] &= !(removed)     # The slice is full if we have removed
       some vectors
161
162     push!(result_slices, (lambda, X, res, res_hist))
163
164     if n_matvec_max < eSlice.n_matvec
165       n_matvec_max = eSlice.n_matvec
166     end
167     converged &= eSlice.converged
168     iterations = max(iterations, eSlice.iterations)
169     end
170   n_matvec += n_matvec_max
171
172   # Initializing merged results
173   @debug "  Merging results"
174   lambda = []
175   X = Array{Float64}(undef, (N, 0))
176   residual_norms = []
177   residual_history = Array{Float64}(undef, (maxiter+1, 0))
178   S_ind = []
179
180   # Comparing neighboring slices to remove duplicates
```

```
for i in 1:ns
  (a, b, m) = S[i]

  # Remove duplicate between slices
  lambda_, X_, res_, res_hist_ = result_slices[i]
  if i > 1 # removing duplicates with the left slice
    lambdal, Xl, resl, res_histl = result_slices[i-1]
    @timeit to "removeDuplicate" lambda_, X_, res_, list, info, removed =
  removeDuplicate2(a, lambdal, Xl, resl, lambda_, X_, res_, [res_hist_];
  return_info=true, right=true)
    res_hist_ = list[]
    info_remove[i] *= "\n Duplicates with slice $(i-1) : "*info
    missing_vect[i] &= !(removed)
  end
  if i < ns # removing duplicates with the right slice
    lambdar, Xr, resr, res_histr = result_slices[i+1]
    @timeit to "removeDuplicate" lambda_, X_, res_, list, info, removed =
  removeDuplicate2(b, lambda_, X_, res_, lambdar, Xr, resr, [res_hist_];
  return_info=true)
    res_hist_ = list[]
    info_remove[i] *= "\n Duplicates with slice $(i+1) : "*info
    missing_vect[i] &= !(removed)
  end

  result_slices[i] = lambda_, X_, res_, res_hist_
  lambda = vcat(lambda, lambda_)
      X = [X X_]
  residual_norms = vcat(residual_norms, res_)
  residual_history = [residual_history res_hist_]
  S_ind = vcat(S_ind, [i for k in 1:length(lambda_)])
end

# Removing extra eigenpairs to keep only nv of them
if length(lambda)<nv # To many vectors removed
  missing_vect[ns] = true
end
@timeit to "removeExtra" lambda, X, list, info_extra, removed =
  removeExtra(H, lambda, X, nv, [residual_norms, residual_history, S_ind];
   return_info=true)
residual_norms, residual_history, S_ind = list

# Sorting the eigenvalues
@timeit to "Sorting eigenvalues" p = sortperm(real(lambda))
lambda = real(lambda[p])
X = X[:, p]
```

```
  ortho = norm(X'*X - I)    # Measure of the orthonormality of X
  residual_norms = residual_norms[p]
  residual_history = residual_history[p]
  S_ind = S_ind[p]

  converged |= !isnothing(n_iterations)

  # Writing logs
  if save_logs
    # removed for this document
    end

    # Updating solving_data
    solving_data[:n_slicing][] += 1
    push!((solving_data[:lambda_history])[i_kpt], lambda)
    push!((solving_data[:X_history])[i_kpt], X)
    push!((solving_data[:residual_history])[i_kpt], residual_norms)
    push!((solving_data[:H_history])[i_kpt], H)
    push!((solving_data[:S_history])[i_kpt], S)
    push!((solving_data[:S_ind_history])[i_kpt], S_ind)
    push!((solving_data[:lambdas_history])[i_kpt], lambdas)
    push!((solving_data[:Xs_history])[i_kpt], Xs)
    push!((solving_data[:residuals_history])[i_kpt], ress)
    push!((solving_data[:degree_history])[i_kpt], degree)
    (solving_data[:missing_vect])[i_kpt] = missing_vect
    (solving_data[:update_slices])[i_kpt] += sum(missing_vect)
    if (solving_data[:update_slices])[i_kpt] > 0  # Les slices not converged
     yet
        (solving_data[:n_conv_slices])[i_kpt] = i_ite
    end
  (; lambda, X, residual_norms, n_matvec, iterations, converged,
    residual_history)
end

function call_slicingEigenSolveur(;
    ns=1,
    filter=ChebyDirac,  # Filter
    argf=Dict(),    # Keyword arguments for the filter
    n_extra=0,
    slices=maxDOSClusteringSlices,
    random=0,
    n_iterations=1,
    n_update=16,
    p=0.1,
    save_logs=true,
```

```
    exact_eigenvalues=false,
    to=DFTK.timer,
    solving_data=init_solving_data_slicing(1) # must be specified
  )
  function f(H, X0; kwargs...)    # kwargs... for DFTK's arguments
    @timeit to "slicingEigenSolver" return slicingEigenSolver(H, X0;
      ns,
      filter,
      argf,
      n_extra,
      slices,
      random,
      n_iterations,
      save_logs,
      exact_eigenvalues,
      n_update,
      p,
      to,
      solving_data,
      kwargs...
    )
  end
  return f
end
```

This custom solver can then be used through the Julia package DFTK as done in the following example.

```
slicing_param_default = Dict(
    :ns => 3,
    :filter=>ChebyDirac,
    :argf=>Dict(:tau=>0.5, :damping=>sigma),
    :n_extra => 10,
    :random => 0.,
    :n_iterations => 1,
    :n_update => 16,
    :p => 0.1,
    :slices => maxDOSClusteringSlices
)

solving_data = init_solving_data_slicing(slicing_param_default[:ns], 1)

scfres_slicing = self_consistent_field(basis;
        tol=1e-6,
        maxiter=20,
        eigensolver=call_slicingEigenSolveur(;
```

```
19              solving_data,
20              slicing_param...
21          )
22  )
```

Bibliography

[BG73] ke Björck and Gene H Golub. Numerical methods for computing angles between linear subspaces. *Mathematics of computation*, 27(123):579–594, 1973.

[Blo29] Felix Bloch. Über die quantenmechanik der elektronen in kristallgittern. *Zeitschrift für physik*, 52(7-8):555–600, 1929.

[Che82] Elliott Ward Cheney. *Introduction to approximation theory*. Chelsea Publishing Company, 1982.

[Els85] Ludwig Elsner. An optimal bound for the spectral variation of two matrices. *Linear algebra and its applications*, 71:77–80, 1985.

[GAA+20] Xavier Gonze, Bernard Amadon, Gabriel Antonius, Frédéric Arnardi, Lucas Baguet, Jean-Michel Beuken, Jordan Bieder, François Bottin, Johann Bouchet, Eric Bousquet, Nils Brouwer, Fabien Bruneval, Guillaume Brunin, Théo Cavignac, Jean-Baptiste Charraud, Wei Chen, Michel Côté, Stefaan Cottenier, Jules Denier, Grégory Geneste, Philippe Ghosez, Matteo Giantomassi, Yannick Gillet, Olivier Gingras, Donald R. Hamann, Geoffroy Hautier, Xu He, Nicole Helbig, Natalie Holzwarth, Yongchao Jia, François Jollet, William Lafargue-Dit-Hauret, Kurt Lejaeghere, Miguel A. L. Marques, Alexandre Martin, Cyril Martins, Henrique P. C. Miranda, Francesco Naccarato, Kristin Persson, Guido Petretto, Valentin Planes, Yann Pouillon, Sergei Prokhorenko, Fabio Ricci, Gian-Marco Rignanese, Aldo H. Romero, Michael Marcus Schmitt, Marc Torrent, Michiel J. van Setten, Benoit Van Troeye, Matthieu J. Verstraete, Gilles Zérah, and Josef W. Zwanziger. The abinit project: Impact, environment and recent developments. *Comput. Phys. Commun.*, 248:107042, 2020.

[GP18] Brendan Gavin and Eric Polizzi. Krylov eigenvalue strategy using the feast algorithm with inexact system solves. *Numerical Linear Algebra with Applications*, 25(5):e2188, 2018.

[Gro23] The ABINIT Group. Abinit, 2023.

[GVL13] GH Golub and CF Van Loan. *Matrix Computation, 3{nd} edition*. Johns Hopkins University Press, Baltimore, MD, 2013.

[GW69] Gene H Golub and John H Welsch. Calculation of gauss quadrature rules. *Mathematics of computation*, 23(106):221–230, 1969.

[HK64] P. Hohenberg and W. Kohn. Inhomogeneous electron gas. *Phys. Rev.*, 136:B864–B871, Nov 1964.

[HLC21] Michael F. Herbst, Antoine Levitt, and Eric Cancès. Dftk: A julian approach for simulating electrons in solids. *Proc. JuliaCon Conf.*, 3:69, 2021.

[Jac30] Dunham Jackson. *The theory of approximation*, volume 11. American Mathematical Soc., 1930.

[JS01] Zhongxiao Jia and GW Stewart. An analysis of the rayleigh-ritz method for approximating eigenspaces. *Mathematics of computation*, 70(234):637–647, 2001.

[KF96] Georg Kresse and Jürgen Furthmüller. Efficient iterative schemes for ab initio total-energy calculations using a plane-wave basis set. *Physical review B*, 54(16):11169, 1996.

[Kny01] Andrew V Knyazev. Toward the optimal preconditioned eigensolver: Locally optimal block preconditioned conjugate gradient method. *SIAM journal on scientific computing*, 23(2):517–541, 2001.

[KR91] L Komzsik and T Rose. Substructuring in msc/nastran for large scale parallel applications. *Computing Systems in Engineering*, 2(2-3):167–173, 1991.

[KS65] W. Kohn and L. J. Sham. Self-consistent equations including exchange and correlation effects. *Phys. Rev.*, 140:A1133–A1138, Nov 1965.

[Lan88] Cornelius Lanczos. *Applied analysis*. Courier Corporation, 1988.

[LSY16] Lin Lin, Yousef Saad, and Chao Yang. Approximating spectral densities of large matrices. *SIAM review*, 58(1):34–65, 2016.

[LT15] Antoine Levitt and Marc Torrent. Parallel eigensolvers in plane-wave density functional theory. *Computer Physics Communications*, 187:98–105, 2015.

[LXES19] Ruipeng Li, Yuanzhe Xi, Lucas Erlandson, and Yousef Saad. The eigenvalues slicing library (evsl): Algorithms, implementation, and software. *SIAM Journal on Scientific Computing*, 41(4):C393–C415, 2019.

[LXV+15] Ruipeng Li, Yuanzhe Xi, Eugene Vecharynski, Chao Yang, and Yousef Saad. A thick-restart lanczos algorithm with polynomial filtering for hermitian eigenvalue problems. *SIAM J. Sci. Comput.*, 38, 2015.

[Pol09] Eric Polizzi. Density-matrix-based algorithm for solving eigenvalue problems. *Physical Review B*, 79(11):115112, 2009.

[PTA+92] Mike C Payne, Michael P Teter, Douglas C Allan, TA Arias, and ad JD Joannopoulos. Iterative minimization techniques for ab initio total-energy calculations: molecular dynamics and conjugate gradients. *Reviews of modern physics*, 64(4):1045, 1992.

[PTP14] Ping Tak Peter Tang and Eric Polizzi. Feast as a subspace iteration eigensolver accelerated by approximate spectral projection. *SIAM Journal on Matrix Analysis and Applications*, 35(2):354–390, 2014.

[Riv81] Theodore J Rivlin. *An introduction to the approximation of functions*. Courier Corporation, 1981.

[Riv20] Theodore J Rivlin. *Chebyshev polynomials*. Courier Dover Publications, 2020.

[Saa11] Yousef Saad. *Numerical methods for large eigenvalue problems: revised edition*. SIAM, 2011.

[Saa16] Yousef Saad. Analysis of subspace iteration for eigenvalue problems with evolving matrices. *SIAM Journal on Matrix Analysis and Applications*, 37(1):103–122, 2016.

[SCS12] Grady Schofield, James R Chelikowsky, and Yousef Saad. A spectrum slicing method for the kohn–sham problem. *Computer Physics Communications*, 183(3):497–505, 2012.

[Woo50] Max A Woodbury. *Inverting modified matrices*. Department of Statistics, Princeton University, 1950.

[WYBY20] David B Williams-Young, Paul G Beckman, and Chao Yang. A shift selection strategy for parallel shift-invert spectrum slicing in symmetric self-consistent eigenvalue computation. *ACM Transactions on Mathematical Software (TOMS)*, 46(4):1–31, 2020.

[XLS18] Yuanzhe Xi, Ruipeng Li, and Yousef Saad. Fast computation of spectral densities for generalized eigenvalue problems. *SIAM Journal on Scientific Computing*, 40(4):A2749–A2773, 2018.

[XS16] Yuanzhe Xi and Yousef Saad. Computing partial spectra with least-squares rational filters. *SIAM Journal on Scientific Computing*, 38(5):A3020–A3045, 2016.

[ZSTC06] Yunkai Zhou, Yousef Saad, Murilo L Tiago, and James R Chelikowsky. Self-consistent-field calculations using chebyshev-filtered subspace iteration. *Journal of Computational Physics*, 219(1):172–184, 2006.