3 | Numerical methods

This chapter is devoted to describing the practical implementation of the nuclear Hartree-Fock method of this work. In section 3.1, the two partial differential equations (PDEs) of interest, the Kohn-Sham equation and the Poisson equation, are numerically approximated through finite differences. In section 3.2.2, a breakdown of numerical solvers for the large-scale eigenvalue problem posed by the KS equation is presented, to pedagogically illustrate the rationale and implementation of the GCG algorithm. Finally, in section 3.3, the self-consistent calculation is presented, along with the implementation of spatial constraints and the optimization of the numerical parameters of GCG.

3.1. Finite differences

The framework used to numerically solve the relevant PDEs of the problem, is the one of finite differences. The core idea is to discretize the domain on a 3D mesh, use Taylor expansions to approximate differential operators and then solve the resulting system of linear equations.

3.1.1. 3D mesh

The first step of the process is representing the different fields in a numerical, discretized fashion. Generally speaking, we deal at most with 2-rank tensors, which depend on three space coordinates and one spin coordinate.
Discretizing the 3D cartesian space with a 3-index mesh, choosing a box whose size along x, y, z is respectively $[-a_x, a_x]$, $[-a_y, a_y]$, $[-a_z, a_z]$, and a number of points n_x, n_y, n_z, the resulting lattice will be given by

$$V = \{(-a_x + ih_x, -a_y + ih_y, -a_z + ih_z)\} = \{(x_i, y_j, z_k)\}$$

Where the indices and step sizes are

$$\begin{aligned} i = 0, \dots, n_x - 1 \quad h_x &= \frac{2a_x}{n_x - 1} \\ j = 0, \dots, n_y - 1 \quad h_y &= \frac{2a_y}{n_y - 1} \\ k = 0, \dots, n_z - 1 \quad h_z &= \frac{2a_z}{n_z - 1} \end{aligned}$$

The following implementation assumes $a = a_x = a_y = a_z = a$ and $n = n_x = n_y = n_z = n$, without losing generality.
Including the spin degree of freedom, we can finally represent the fields in a numerical way through

$$\varphi(\boldsymbol{r}, \sigma) \mapsto \varphi(x_i, y_j, z_k, s) = \varphi_{ijks} \tag{3.1}$$

Differential operators discretization

By using Taylor series, it's possible to write approximations to derivatives [45], in any point of the lattice, of any (reasonable) order of accuracy, involving only near neighbouring points. In the present work, 5-points derivatives are used, meaning Taylor expansions are written for $\varphi(x \pm h)$ and $\varphi(x \pm 2h)$ to compute the differential operators. Formulae for first and second derivates are given in appendix A.2.
From the theory background of chapter 2, we discern two main kinds of PDEs: the Schrödinger-like KS equation, and the Poisson equation.

3.1.2. Schrödinger equation

Starting from the Schrödinger equation (2.76), reported here for clarity

$$\left[-\nabla\left(\frac{\hbar^2}{2m_q^*(\mathbf{r})}\nabla\right) + U_q(\mathbf{r}) + \delta_{\mathrm{q,proton}} U_C(\mathbf{r}) - i\mathbf{B}_q(\mathbf{r}) \cdot (\nabla \times \boldsymbol{\sigma}) \right] \varphi = \varepsilon\varphi$$

it can be compactly written as

$$f(\nabla^2\varphi, \nabla\varphi, \varphi, \boldsymbol{r}, s) = \varepsilon\varphi. \tag{3.2}$$

If f is linear in φ, it is be possible to rewrite it as a linear combination of the values of φ on the mesh, after which we can use linear algebra methods to solve the problem.

Linearity Breaking down each part of the equation, the kinetic term

$$\nabla\left(\frac{\hbar^2}{2m_q^*(\mathbf{r})}\nabla\right)\varphi = \frac{\hbar^2}{2m_q^*(\mathbf{r})}\nabla^2\varphi + \nabla\left(\frac{\hbar^2}{2m_q^*(\mathbf{r})}\right)\cdot\nabla\varphi \tag{3.3}$$

is evidently linear in φ.
The spin-orbit term of (2.76), which we write as

$$\begin{aligned}\hat{h}_{\mathrm{SO}} &= -i\mathbf{B}_q(\boldsymbol{r})\cdot(\nabla\times\boldsymbol{\sigma}) \\ &= -i[\mathbf{B}_{q,x}(\boldsymbol{r})(\sigma_z\partial_y - \sigma_y\partial_z) + \mathbf{B}_{q,y}(\boldsymbol{r})(\sigma_x\partial_z - \sigma_z\partial_x) + \mathbf{B}_{q,z}(\boldsymbol{r})(\sigma_y\partial_x - \sigma_x\partial_y)]\end{aligned}$$

is also linear in φ.
Finally, the mean field terms U_q, U_c

$$(U_q + \delta_{\mathrm{q,proton}}U_c)\varphi$$

are just multiplicative, hence linear.
Given that the whole equation is linear in φ, we can evaluate it on the chosen mesh, using finite differences to approximate the differential operators, yielding a linear eigenvalue problem of the form

$$\sum_{\alpha=1}^{N} A_{\alpha\beta}\varphi_\beta = E\varphi_\alpha \tag{3.4}$$

where the shorthand notation $N = 2\cdot N_x\cdot N_y\cdot N_z$ is used to denote the size of the matrix A, which is $N\times N$.

Boundary conditions

We expect the nucleus to be a localized object, leading to vanishing Dirichlet boundary conditions for the Schrödinger equation. Near the boundaries, the derivatives will involve points outside the box and setting these points to zero, is equivalent to solving

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & A & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \varphi_{-2} \\ \varphi_{-1} \\ \varphi \\ \varphi_N \\ \varphi_{N+1} \end{bmatrix} = E \begin{bmatrix} \varphi_{-2} \\ \varphi_{-1} \\ \varphi \\ \varphi_N \\ \varphi_{N+1} \end{bmatrix} \tag{3.5}$$

From this system of equations, we get for points outside the boundary:

$$\begin{cases} \varphi_{-2} = 0 \\ \varphi_{-1} = 0 \\ \dots \end{cases} \tag{3.6}$$

Meaning that φ outside the box will automatically be set to zero if the A matrix is built assuming those points to be zero when computing its coefficients.

3.1.3. Poisson equation

The other fundamental PDE we need to solve is the Poisson equation encountered in section (REF). Dropping the c and p subscripts, it reads

$$\nabla^2 V = 4\pi e^2 \rho$$

It's simpler than the Schrödinger equation, as it only involves a Laplacian and it's not an eigenvalue problem. The right is side is given, and the solution is found by inverting the coefficients' matrix.

Boundary conditions

Unlike the Schrödinger equation, we do not expect the solution to rapidly decay near the boundaries; as reported in section 2.4, we have fixed, non-vanishing boundary conditions, which we have to properly impose on the system.
We can choose a direction, say x, and look at the discretized equation at the boundaries $x = \pm a$. Since the indices j, k won't vary, we can omit them, and ignore the other derivates in the following equations.

$$\begin{aligned} \nabla^2 V &= \partial_{xx} V + \partial_{yy} V + \partial_{zz} V \\ &= \frac{-V_{i-2} + 16V_{i-1} - 30V_i + 16V_{i+1} - V_{i+2}}{12h^2} + \dots = 4\pi e^2 \rho_i \end{aligned} \tag{3.7}$$

Near a boundary, say $i = 0$, the formula calls for points outside the box, known as *ghost points*. Since they are not part of the linear system, but they are known, we can bring them on the right side of equation (3.7).

$$\frac{-30V_0 + 16V_1 - V_2}{12h^2} = 4\pi e^2 \rho_0 + \frac{V_{-2} - 16V_{-1}}{12h^2} = \tilde{\rho}_0 \tag{3.8}$$

The same procedure must be applied to all equations involving ghost points, e.g. for $i = 1$

$$\frac{+16V_0 - 30V_1 + 16V_2 - V_3}{12h^2} = 4\pi e^2\rho_1 + \frac{V_{-1}}{12h^2} = \tilde{\rho}_1. \tag{3.9}$$

The proper system to solve will then be

$$AV = \tilde{\rho} \tag{3.10}$$

Where A is constructed as previously specified. Solving with $\tilde{\rho}$ on the right hand side will force the solution to obey boundary conditions.

On higher order approximations and performance

Higher and higher order approximations for derivatives involve more points that are further away. This increases accuracy by reducing the finite differences error, but it also decreases the matrix sparseness.

3.2. Eigenvalue problem

This section is devoted to the approximate solution of the eigenvalue problem, needed for the Schrödinger equation (2.76).
Eigenvalue problems are ubiquitous in physics and engineering, and while solving one for a small matrix is trivial, it still requires roughly $O(n^3)$ operations [17] to do so. More often than not, real computational applications result in large-scale matrices, which are completely out of question for exact eigenvalues calculations, thus requiring the use of approximate algorithms.
We will begin by describing common building blocks of iterative eigensolvers in section 3.2.1, namely:

- the approximate solutions of linear systems by the use of the Conjugate Gradient method;
- matrix preconditioning to speed up convergence;
- the Rayleigh-Ritz procedure to find good approximations to the eigenpairs in a certain subspace; and
- the shift-and-invert method, to select the desired portion of the eigenvalue spectrum.

After describing these building blocks, some of the most commonly used eigensolvers are described in section 3.2.2, focusing on the core ideas and stating their limitations, to

finally address the General Conjugate Gradient method, whose implementation in the present work is detailed in section 3.2.3.

3.2.1. Conjugate Gradient and numerical techniques

Conjugate Gradient method

Solving linear systems of the form

$$Ax = b \tag{3.11}$$

is crucial in many eigensolvers. The Conjugate Gradient (CG) is perhaps the most famous iterative solver in this sense, especially in regards to sparse matrices, as we will see in a moment. CG applies to cases where A is a real, $n \times n$, positive-definite, symmetric matrix, and x and b are n-dimensional vectors.
Many generalizations to this method exist, which relax the requirements on the matrix, like BiCGSTAB, CGRES and so on [36]. We will describe the working principle of CG, but the same applies to all the others, with slight variations.

Steepest descent method The quadratic form $f(x)$ derived from the system (3.11) is

$$f(x) = \frac{1}{2}x^T Ax - b^T x \tag{3.12}$$

If A is symmetric, positive-definite, the shape of $f(x)$ is convex and has a global minimum for

$$\nabla_x f(x) = Ax_m - b = 0 \implies Ax_m = b, \tag{3.13}$$

hence the extremum of the quadratic form is the also the solution of the linear system (3.11).
We can employ the well-known gradient descent technique [37] to find such point: starting from a guess x_0, we compute the direction where f decreases the most (the residual r_i), compute the step size for maximal decrease, and update x_i at each iteration accordingly, repeating until convergence.

$$d_i = r_i = = b - Ax_i \tag{3.14}$$

$$x_{i+1} = x_i + \alpha_i r_i \tag{3.15}$$

$$\text{with } \alpha_i \text{ such that } \frac{\mathrm{d}f}{\mathrm{d}\alpha_i} = 0 \implies \alpha_i = \frac{r_i^T r_i}{r_i^T A r_i} \tag{3.16}$$

This is a powerful but highly inefficient procedure. We are not ensuring that the search direction doesn't end up with components in subspaces that were explored already.
It can be proven [37] that the norm of the error $e_i = x_i - x_m$ is minimal at each iteration if the search directions d_i are chosen to be A-orthogonal to the next error, i.e. $d_i^T A e_{i+1} = 0$. This makes the algorithm converge at the exact solution in n steps, but most importantly it allows to truncate the iterations without a large error on the approximation x_i.
In this case, the algorithm is called Conjugate Gradient Method and is formulated as

$$\alpha_i = \frac{r_i^T r_i}{d_i^T A d_i} \tag{3.17}$$

$$x_{i+1} = x_i + \alpha_i d_i \tag{3.18}$$

$$r_{i+1} = r_i - \alpha_i A d_i \tag{3.19}$$

$$\beta_{i+1} = \frac{r_{i+1}^T r_{i+1}}{r_i^T r_i} \tag{3.20}$$

$$d_{i+1} = r_{i+1} + \beta_{i+1} d_i \tag{3.21}$$

where iterations are truncated if the norm of the residual r_i is smaller than a certain threshold. CG represents a great method for sparse matrices, because it can be proven to be of complexity $O(m)$, where m is the number of non-zero elements in A [37].

Complex matrices Algorithm (3.17) and the CG method in general can be used for complex matrices, under the condition that A is Hermitian and positive-definite when using the complex inner product, meaning that

$$A = A^\dagger \text{ and } x^\dagger A x > 0. \tag{3.22}$$

Preconditioning

The CG method convergence is known to be limited by the modulus of the condition number of A $\kappa(A)$, given by [37]

$$\kappa(A) = \frac{\lambda_{\max}(A)}{\lambda_{\min}(A)} \tag{3.23}$$

where $\lambda_{\max}$ and $\lambda_{\min}$ are respectively the largest and smallest eigenvalues of A in magnitude. If we were able to find a good *preconditioner* M, symmetric and positive-definite, such that $\kappa(M^{-1}A) \lll \kappa(A)$, and M^{-1} is easy to compute, then the algorithm would

converge much faster, by solving $M^{-1}Ax = M^{-1}b$, since x is also the solution of $Ax = b$.

$$x = (M^{-1}A)^{-1}M^{-1}b = A^{-1}MM^{-1}b = A^{-1}b. \tag{3.24}$$

Without delving into the details of the preconditioner implementation, detailed in [37], note that, in general, $M^{-1}A$ is neither positive-definite nor symmetric, which requires a Cholesky decomposition [24] $M = EE^T$ to be used, so that the problem may be restated with a symmetric positive-definite matrix $E^{-1}AE^{-T}$.
The catch with preconditioning is that M has no unique recipe. Preconditioners are widely spread across numerical analysis, so many methods have been explored and implemented [32].

Rayleigh-Ritz procedure

A common denominator of all these algorithms is the search of good approximations for the correct eigenvectors in a certain subspace. The method is called Rayleigh-Ritz (RR) procedure [36], and is here outlined.
Let us suppose to have a matrix A of size $n \times n$, with entries in $\mathbf{C}$ and a collection of vectors k organized in a matrix K, where K is of size $n \times k$. Generally speaking, n is large, while k is much smaller.
The best approximation of the true eigenvectors of A in the subspace $\mathcal{K}$ spanned by the vectors in K can be computed by solving the small scale eigenvalue problem

$$K^\dagger AKC = C\Lambda. \tag{3.25}$$

Here matrices $K^\dagger AK$ and C are of size $k \times k$. Computing KC gives a matrix of size $n \times k$, whose column vectors are the best approximations of the true eigenvectors of A in the subspace $\mathcal{K}$, with their corresponding eigenvalues in the entries of the diagonal matrix Λ.

Shift and Invert

The power iteration is the technique on which Krylov subspace search methods are based [17]. By repeatedly applying matrix A to a vector x, x gets skewed towards the eigenvector whose eigenvalue is of largest magnitude λ_n.
Assume A is a hermitian matrix, thus diagonalizable. This means we can write an arbitrary vector $x^{(0)}$ as a linear combination of the eigenvectors $\{v_i\}$ of A.

$$x^{(0)} = \sum_i^n \alpha_i v_i \tag{3.26}$$

If we apply A to $x^{(0)}$ k times, we get

$$x^{(k)} = A^k x^{(0)} = \sum_i^n \alpha_i A^k v_i = \sum_i^n \alpha_i \lambda_i^k v_i \tag{3.27}$$

It can be proven that the ratio of the j-th component of $x_j^{(k)}$ and $x_j^{(k-1)}$ converges to λ_n

$$\lim_{k\to\infty} \frac{x_j^{(k)}}{x_j^{(k-1)}} = \lambda_n \tag{3.28}$$

which means, that for large enough k, we have the relation

$$Ax^{(k)} \approx \lambda_n x^{(k)} \tag{3.29}$$

So $x^{(k)}$ is an approximation of the eigenvector v_n of A whose eigenvalue is λ_n.

Smallest eigenvalue If instead of the largest eigenvalue, we were interested in the smallest one (in magnitude) λ_0, then we would need to apply the inverse matrix A^{-1} to $x^{(k)}$, which would change the ratio (3.28) to

$$\lim_{k\to\infty} \frac{x_j^{(k)}}{x_j^{(k-1)}} = \lambda_0 \tag{3.30}$$

Let us assume for a moment that we're solving a nuclear single-particle Hamiltonian, where we have a certain number of bound states of negative energy and a much larger number of unbound states with positive energy. In this case, the inverse power iteration would converge to the states whose energy is closer to zero, avoiding the interesting ones on the bottom of the spectrum.
The solution is, before inverting, to shift the matrix by a quantity σ that is very close to the lowest eigenvalue we want to compute, call it λ_σ (eigenvector v_σ). Now, the eigenvalue of lowest magnitude of $(A - \sigma I)$ is $\lambda_\sigma - \sigma$ and by applying $(A - \sigma I)^{-1}$ to $x^{(k)}$, we will get the approximation to the eigenvector v_σ.

3.2.2. Iterative eigensolvers

Now that the main techniques used by iterative eigensolvers have been laid out, we can look at three general methods, which are the most commonly used ones.

Jacobi-Davidson

The Jacobi-Davidson, where at each iteration by a correction to the previous eigenvectors. Given an approximation (u, θ) of an eigenpair of matrix A, where u is the approximate eigenvector and θ is the approximate eigenvalue, if the residual

$$r = Au - \theta u \tag{3.31}$$

is ≈ 0, then the eigenpair converged. Otherwise, we want to find a correction t such that

$$r = A(u + t) - (\theta + \delta\theta)(u + t) = 0 \tag{3.32}$$

Linearizing this equation in t gives

$$(A - \theta I)t = -r \tag{3.33}$$

To avoid singularity of the equation near convergence, since u approximately spans a subspace of the system's kernel $\ker(A - \theta I)$, and enrich the subspace search with a useful orthogonal correction, we project the problem onto the orthogonal subspace of u, which finally gives

$$(I - uu^\dagger)(A - \theta I)(I - uu^\dagger)t = -r \tag{3.34}$$

Algorithm 3.1 Jacobi-Davidson method

1: Choose normalized initial vectors $\{u_k\}$, set $V = [u_1, \ldots, u_k]$
2: **repeat**
3: Compute Ritz pair: $T = V^\dagger AV$, solve $Ty = \theta y$
4: Set $u = Vy$, residual $r = Au - \theta u$
5: **if** $\|r_k\| < \varepsilon \ \forall k$ **then**
6: **return** (θ, u)
7: **end if**
8: Solve approximately $(I - u_k u_k^\dagger)(A - \theta I)(I - u_k u_k^\dagger)t_k = -r_k$ using preconditioned iterative solver, ensuring $t_k \perp u_k$
9: Normalize: $v_k = t_k / \|t_k\|$
10: Expand subspace, setting $V = [V, v]$
11: **until** convergence for $k = 1, \ldots,$ nev

Although simple, this method is computationally efficient only by using preconditioning,

which is known to be unstable in many cases [36].

Lanczos

Lanczos algorithm [25] is probably the most used iterative eigensolver in regards to hermitian matrices. It's a Krylov subspace search method, meaning the Rayleigh-Ritz procedure is done on a subspace formed as

$$\mathcal{K} = \{v_1, Av_1, A^2v_1, \ldots, A^{k-1}v_1\} \tag{3.35}$$

which exploits the power iteration. After orthogonalizing the new approximation to the previous one and diagonalizing the small scale problem, we end up with the new best approximations to the eigenvectors of A.

Algorithm 3.2 Lanczos Method

Choose normalized initial vector v_1, set $\beta_0 = 0$, $m =$ subspace size.
repeat
  for $j = 1, 2, \ldots, m$ **do**
    $w \leftarrow Av_j - \beta_{j-1}v_{j-1}$
    $\alpha_j \leftarrow v_j^* w$
    $w \leftarrow w - \alpha_j v_j$
    $\beta_j \leftarrow \|w\|$
    if $\beta_j = 0$ **then**
      break
    end if
    $v_{j+1} \leftarrow w/\beta_j$
  end for
  Form tridiagonal matrix $T_m = \text{tridiag}(\beta_{1:m-1}, \alpha_{1:m}, \beta_{1:m-1})$
  Compute eigen-decomposition $T_m y_k = \theta_k y_k$, for $k = 1, \ldots,$ nev
  Form Ritz approximations $x_k = V_m y_k$, where $V_m = [v_1, \ldots, v_m]$
  Compute residual norms $r_k = \|Ax_k - \theta_k x_k\|$ for all k
until convergence for $k = 1, \ldots,$ nev

Lanczos is extremely efficient, memory- and CPU-wise for extremal eigenvalues, but this limits its applicability, as one may be interested in the inner portion of the eigenvalue spectrum, such in the case of Hartree-Fock-Bogoliubov (HFB).
A shift-and-invert strategy would be unfeasible in the case of large scale problems, since all Lanczos steps need to be performed exactly to avoid instabilities, a well known problem

in the Arnoldi generalization [36].

LOBPCG

The last algorithm of this short list is LOBPCG, it's the newest and most sophisticated one of the three.
Introduced by A. V. Knyazev in 1991 [22], it's a block, preconditioned conjugate gradient method, explicitly targeted at solving large-scale eigenvalue problems, and it has been used in modern solutions of the Schrödinger/KS equation in recent years [26, 28, 31, 44]. We won't go into the details of LOBPCG, since GCG shares with it many aspects, like blocking and search directions calculation.
LOBPCG works very well for large scale problems, but it has limitations. For one, it's not possible to arbitrarily select the portion of the matrix spectrum to calculate, which is required for problems where variational collapse happens, like in HFB or the Dirac equation, which manifests particle/antiparticle solutions [26]. To solve this, an additional filtering step is required [26, 28], which introduces a computational cost in the algorithm. Lastly, LOBPCG may fail when poor conditioning is present or when high precision on the eigenvalues is required [27].

3.2.3. General Conjugate Gradient

The General Conjugate Gradient is an iterative eigensolver designed with the aim of improving LOBPCG, it is a blocked algorithm, which uses the inverse power method and previous search directions to generate the search subspace. GCG is proven to be faster and more stable than LOBPCG [27].
The search subspace is built as

$$V = [X, P, W], \tag{3.36}$$

where X is the matrix containing the approximations to the eigenvectors of matrix A, P is the matrix containing the previous search directions, and W is the matrix containing the eigenvectors on which the inverse power method is applied approximately using CG. A slightly different implementation of the algorithm is employed in the present work, detailed in algorithm 3.3, to improve applicability to HF calculations and reduce the computational cost.

Eigenvalue problem The original algorithm aims at solving the general eigenvalue problem $AX = \lambda BX$. Since in our case $B = I$, it is omitted from the procedure, reducing the computational cost of the algorithm, in particular, the one of the search direction

block P. After orthonormalization of V, the columns of X are orthonormal as well and the calculation of P is given by

$$P = X_{\text{new}} - X(X^{\dagger}X_{\text{new}}) \tag{3.37}$$

which is the projection of X_{new} onto the orthogonal complement of X.

Complex matrix The algorithm has been generalized to the complex case, where the matrix is complex Hermitian, as such, the transposition operation is replaced by the conjugate transpose.

Blocking The algorithm is designed to allow blocking of the eigenvectors, such that $X = [X_c, X_a, X_r]$, where X_c are the converged eigenvectors, X_a are the active eigenvectors on which we perform the inverse power iteration, with $\text{col}(X_a)$ being a fixed number, and X_r are the remaining eigenvectors, which are to be inserted in X_a as soon as some of its columns converge. This allows to save some computations by avoiding the expensive inverse power on pairs that have already converged. Since in a self-consistent calculation the matrix changes rapidly and at each HF iteration, it is the case that the maximum number of iterations is reached before convergence of all eigenpairs, so we must work at all times on the remaining unconverged eigenvectors. For this reason, we only implement the $X = [X_c, X_a]$ scheme.

Orthogonalization The original paper [27] suggests an improved orthogonalization procedure; being beyond the scope of this work, the simpler Gram-Schmidt [6] orthogonalization is used in the present work.

Shift update The shift update is either fixed, in case of known spectrum, eg for HFB $\texttt{shift} = 0$, or adaptive [27], so that the inverse power step can find the lowest eigenvalues, using the update formula

$$\texttt{shift} = (\lambda_{\text{nev}} - 100\lambda_1)/99 \tag{3.38}$$

where λ_{nev} is the biggest eigenvalue of the RR procedure and λ_1 is the smallest of the active eigenpairs.

Preconditioning The use of a preconditioner is beyond the scope of this work, a simple diagonal preconditioner is used.

Algorithm 3.3 General Conjugate Gradient algorithm

Input: Matrix A, number of desired eigenpairs `nev`, X_{guess} initial guess with $\text{col}(X_{\text{guess}}) = k \geq \text{nev}$, `max_iter` maximum number of iterations.
Initialize block $X = [X_{\text{c}}, X_{\text{a}}] \leftarrow X_{\text{guess}}$
Initialize blocks P and W with k null vectors
Solve the Rayleigh Ritz problem $X^{\dagger}AXC = C\Lambda$
Update $X = XC$
Initialize `shift`, Initialize `iter` $= 0$
while $\text{col}(X_c) <$ `nev` and `iter` $<$ `max_iter` **do**
    Solve approximately $(A + \texttt{shift} \cdot I)W = X_a\Lambda$ with some CG steps, initial value X_a to generate W
    Orthogonalize $V = [X, P, W]$
    Solve the Rayleigh Ritz problem $V^{\dagger}(A + \texttt{shift} \cdot I)VC = C\Lambda$
    Update $X_{\text{new}} = (VC)(:, \texttt{nev})$ and $\Lambda_{\text{new}} = \Lambda - \texttt{shift} \cdot I$
    Compute $P = X_{\text{new}} - X(X^{\dagger}X_{\text{new}})$
    Compute the residual $R = AX_{\text{new}} - \Lambda X_{\text{new}}$
    Check convergence on k-th column norm of R, if $\|R(:,k)\| <$ `tol`, move $X_a(:,k)$ to X_c.
    Update `shift` and `iter`
end while
Output: Approximate eigenpairs (Λ, X)

3.3. Code implementation details

In this last section regarding numerical methods, some important features about the actual code implementation of the HF method are discussed. Mainly, the implementation of the Augmented Lagrangian Method to enforce spatial constraints on the HF solution, the pseudocode of the entire self-consistent procedure, and the choice of optimal parameters for the functional minimization.

3.3.1. Constraints

The purpose of spatial constraints is to find the minimum of the energy functional under the condition that the expectation value of a given operator $\mathcal{Q}$ equals a prescribed target value q_0. Constrained calculations are a fundamental tool to assess the stability of the ground-state minimum and to investigate dynamical properties of the nucleus, such as fission barriers [7].

Constraints can be formulated as an equality-constrained optimization problem (ECP), formulated as

$$\min_{|\Psi\rangle} E \tag{3.39}$$

$$\text{constrainted to } \langle\Psi|\,\mathcal{Q}\,|\Psi\rangle = \langle\mathcal{Q}\rangle = q_0 \tag{3.40}$$

Which yields the Lagrangian

$$E' = E + \lambda(\langle\mathcal{Q}\rangle - q_0) \tag{3.41}$$

where λ is a Lagrange multiplier determined by the condition $\langle\mathcal{Q}\rangle = q_0$. After finding the minimum of E', it's trivial to show that for a given λ, we get [15]

$$\frac{\mathrm{d}E}{\mathrm{d}\,\langle\mathcal{Q}\rangle} = -\lambda. \tag{3.42}$$

From a numerical standpoint, λ needs to be tuned at each iteration to reach the desired value of q_0. This method was the one used in early constrained Hartree-Fock calculations [13].
Although this method is simple, it often fails. Moreover, for the same value of λ, many, possibly infinite values of $\langle\mathcal{Q}\rangle$ can be obtained, for which we are only allowed to get the one with the most stable solution.
A different method is provided by the Quadratic Penalty Method (QPM). Briefly speaking, instead of a Lagrange multiplier, we add a quadratic contribution to the functional, such that

$$E' = E + \frac{c}{2}(\langle\mathcal{Q}\rangle - q_0)^2. \tag{3.43}$$

This is a straightforward method; intuitively one penalizes (hence the name), any solution for which $\langle\mathcal{Q}\rangle \neq q_0$ by increasing its energy. However, the success of such procedure is heavily influenced by the choice of c, often leading to instabilities for large values.
What happens is that for small values of c, the penalty may be insufficient to reach the target q_0, while for bigger values, the penalty may be so strong that the self-consistent calculation oscillates and fails.

Augmented Lagrangian Method

A modern, robust approach, used by HF/HFB codes [10, 35] is given in the form of the Augmented Lagrangian Method (ALM) [39]. Its main idea is to combine the precision of the ECP with the accuracy of the QPM.
Without delving into cumbersome mathematical details, we'll see how the algorithm is practically implemented in the code.

Given the functional

$$E' = E + \lambda(q - q_0) + \frac{c}{2}(q - q_0)^2 \tag{3.44}$$

where $q = \langle \mathcal{Q} \rangle$, the resulting mean field potential will be given by

$$U' = U + \lambda \mathcal{Q} + c(q - q_0)\mathcal{Q} \tag{3.45}$$

$$= U + c(q - q_0(\lambda))\mathcal{Q} \tag{3.46}$$

where $q_0(\lambda)$ is updated at each iteration with the formula

$$q_0(\lambda) = q_0 - \frac{\lambda}{c} \tag{3.47}$$

$$\lambda^{(i+1)} = \lambda^{(i)} + \mu c(q - q_0) \tag{3.48}$$

Here, a slight deviation from the original ALM is present. Since the original work [39] doesn't provide guidance regarding what is considered an *iteration*, we employ the strategy [10] of using a damping factor $\mu \in [0, 1]$, so λ can be updated at each HF iteration for fast convergence, without large oscillations or instabilities of any kind.
This method is what powers the deformation curves that are shown in section (REF), allowing to explore the energy surface with arbitrary precision in reaching the value of q_0 at convergence, provided that enough HF iterations are performed.
Note that, since $\lambda^{(0)} = 0$, for $\mu = 0$ ALM reduces to the standard QPM.

3.3.2. Details on the implementation of the code

The whole Hartree-Fock framework presented up to this point, has been implemented using the C++ language [41] and the Eigen linear algebra library [19], which implements linear algebra operatrions through low level routines such as `LAPACK` and `BLAS`. In figure 3.1, the schematics of the program structure is reported.

3.3.3. Optimal parameters choice

Inside the 'Diagonalize h' step in figure 3.1, the execution of the GCG algorithm is performed, using the current iteration's single-particle Hamiltonian as the matrix to diagonalize and the previous iteration's orbitals as the initial guess. This is the main computational bottleneck of the code, where a correct choice of the execution parameters can drastically reduce execution times. The paramters that need to be chosen carefully are essentially the inverse power step tolerance and the number of maximum GCG iterations.

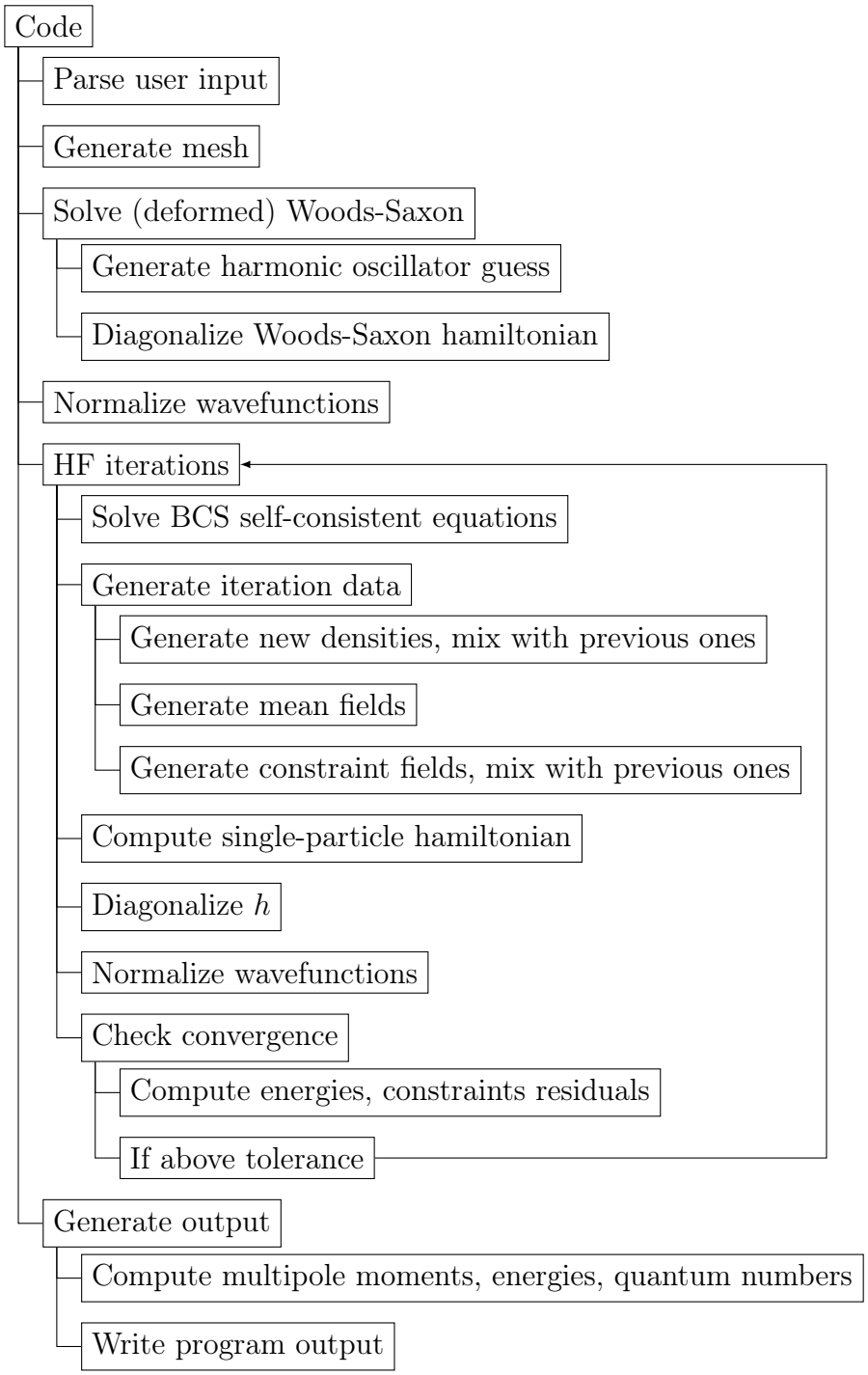

Figure 3.1: Pseudocode of the Hartree-Fock program.

Inverse power step tolerance

The first parameter to be tuned is the tolerance on the CG step which approximately solves the system

$$(A + \texttt{shift} \cdot I)W = X_a\Lambda \tag{3.49}$$

in algorithm 3.3, where A is actually the single-particle Hamiltonian h. When the CG residual $(A + \texttt{shift} \cdot I)W - X_a\Lambda$ is smaller than the tolerance, the procedure stops and outputs the W block.
In figure 3.2, the relative absolute error of the total energy is calculated against a reference benchmark value (details in the results chapter 4), for different values of the CG tolerance. It's clear that at least a tolerance of 10^{-3} is needed for good convergence, while tolerances $\geq 10^{-4}$ stop offering increasing returns, rendering a choice between 10^{-4} and 10^{-5} an optimal one.

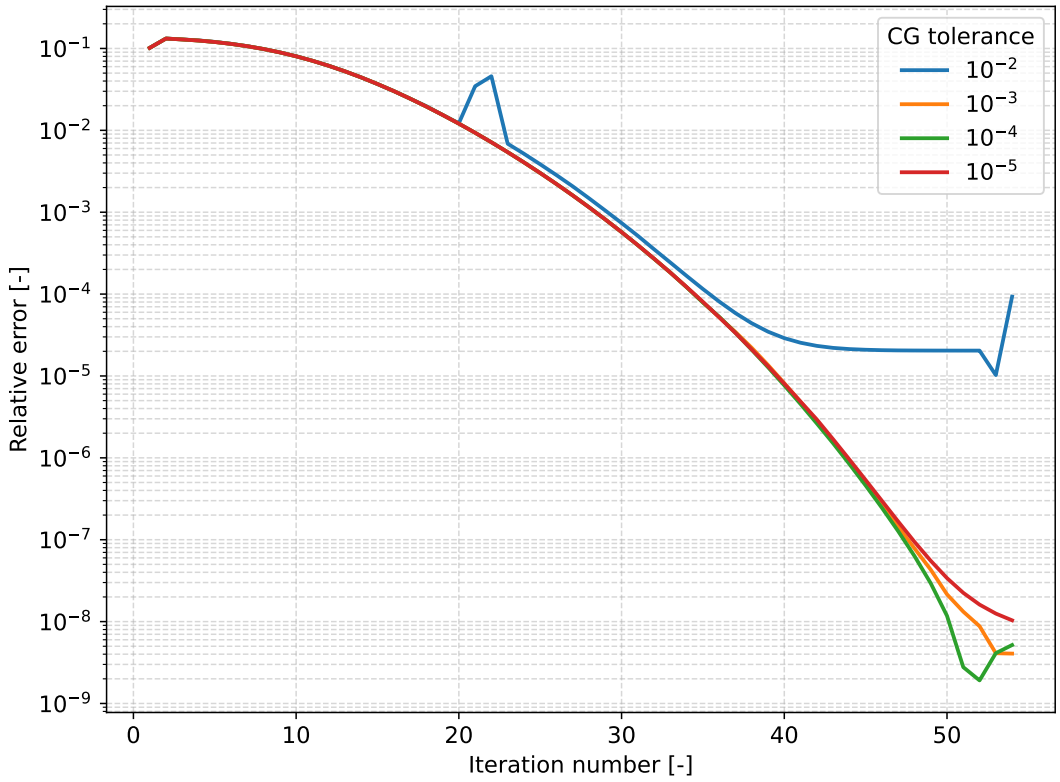

Figure 3.2: HF calculation convergence with varying CG tolerance for ^{16}O, box $[-9, 9]$ fm, step size 0.3 fm.

Inner GCG iterations

The number of inner GCG maximum iterations, here named 'inverse power steps' to avoid confusion, is slightly more nuanced than the CG tolerance. The algorithm converges to the true eigenpairs as the power steps are performed, so one could think that a higher number of steps would bring to HF convergence faster, since the precision on the eigenvalues increases, but this is not the case.
In figures 3.3 and 3.4, the convergence of the HF calculation is plotted for different number

of steps, respectively, for the spherical nucleus ^{16}O and the deformed nucleus ^{24}Mg.

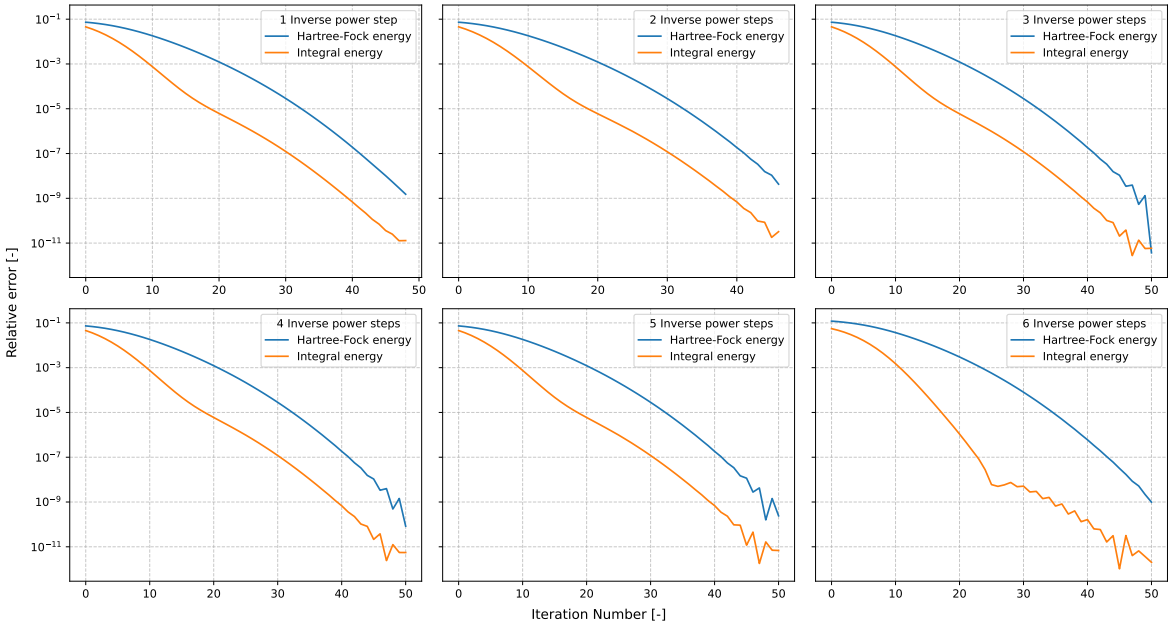

Figure 3.3: HF calculation convergence with varying number of inverse power steps for ^{16}O, box $[-9, 9]$ fm, step size 0.3 fm.

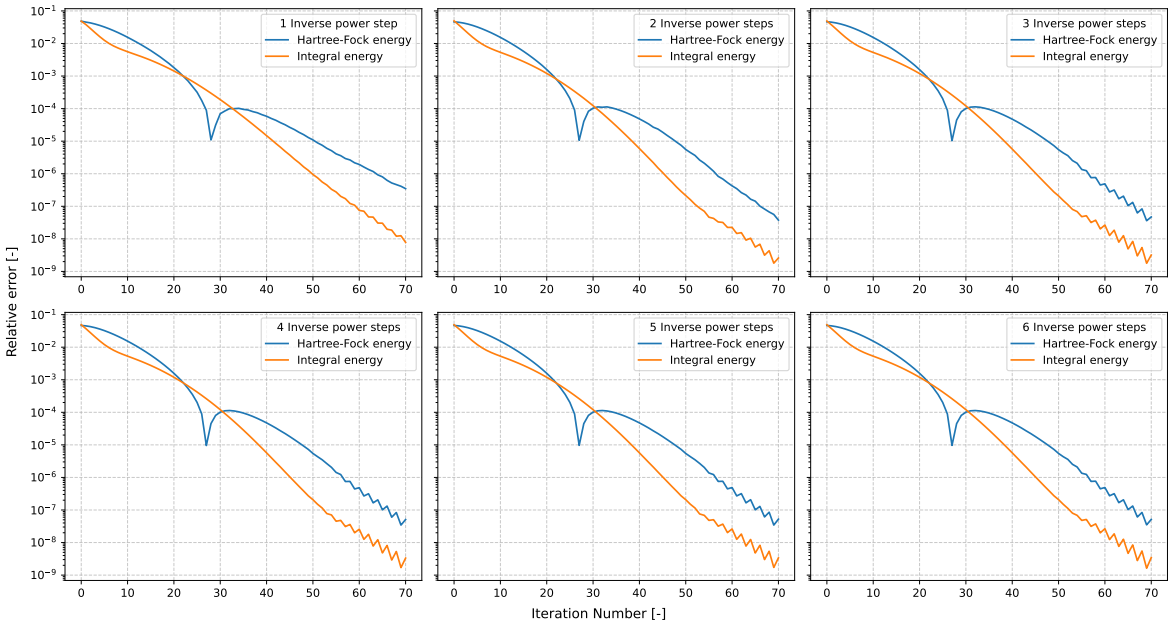

Figure 3.4: HF calculation convergence with varying number of inverse power steps for the deformed nucleus ^{24}Mg, box $[-10, 10]$ fm, step size 0.33 fm.

It's evident that in both cases, a steps number greater than 3 leads to oscillating behaviour

near convergence, without accelerating it, while in the case of the spherical nucleus, just one step is enough to quickly, and reliably reach convergence. In any case, it's clear that delaying the inverse power steps to later HF iterations is safer in terms of stability.
This counter intuitive behavior is likely due to the fact that at each HF iteration the hamiltonian changes and a great number of steps leads to solutions too biased towards the current matrix eigenpairs, at the expense of the next iteration; however, in the case of deformed nuclei, due to sharp shape changes at the start of the calculation, just one step may not be enough to sustain the pace at which the Hamiltonian changes, hence the quicker convergence with more steps.

3.3.4. Numerical stability

As a final remark, the numerical stability of the solver is reported in figure 3.5. The map is produced for a spherical calculation of ^{16}O, with varying box and mesh sizes.
It's possible to observe that for a box whose side is at least ≈ 2.5 times the nuclear radius, the solver numerical stability is loosely dependent on the box extension, but rather on the step size. This is not surprising, as the points separation in space h dictates the precision of the discretized derivatives, as mentioned in section 3.1.

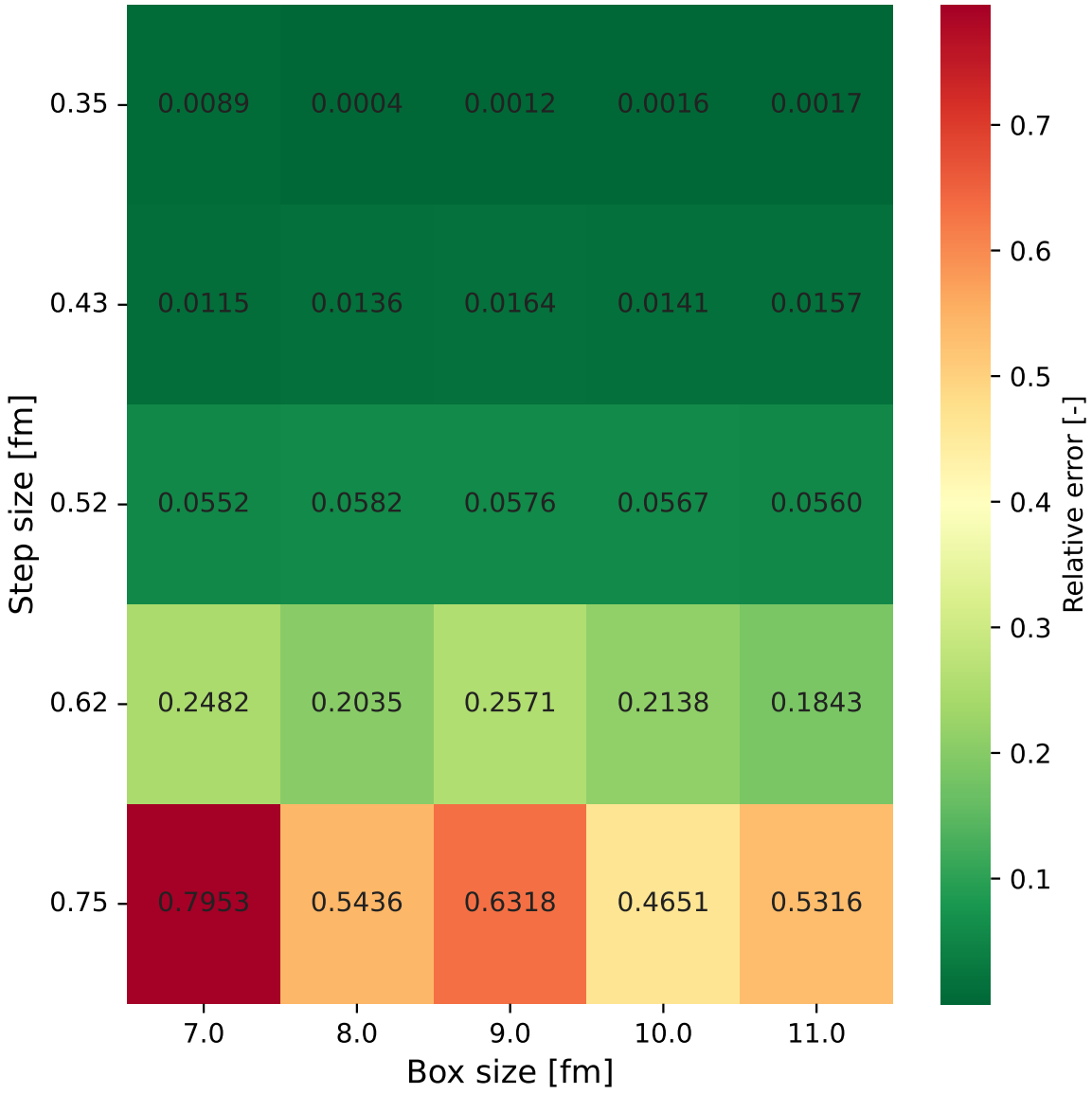

Figure 3.5: Numerical stability map of the HF solver for ^{16}O for different box and step sizes. Relative error is taken against a benchmark reference value.