Quantum mechanics on a computer

Version 1.06 — September 2019 — Phil Hasnip

Contents

1. Introduction

In many electronic structure calculations we need to solve the time-independent Schrödinger equation,

$$H\psi_b = E_b\psi_b$$

where H is the Hamiltonian, ψ_b is a wavefunction (for electron b) and E_b is its energy. This is a large eigenvalue problem, but usually we're only interested in the states with the lowest energies. Because of the Pauli exclusion principle, no two electrons can be in the same state so we need to get enough eigenstates to accommodate all of the electrons.

In this Practical we will concentrate on a class of eigenvalue problems that arises in plane-wave density functional theory. For any particular electron, the wavefunction $\psi(r)$ is expanded in a basis set of plane-waves $e^{iG.r}$ (i.e. a Fourier basis), which enables efficient computation of all of the terms in the Hamiltonian matrix H.

$$\psi(r) = \sum c_G e^{iG.r}$$

where r is the position in space, and each plane-wave has a certain spatial frequency G called the wavevector; the larger $|G|$ is, the shorter the wavelngth and the faster the wave oscillates in space. The coefficients c_G are complex, and determine how much of each plane-wave component the wavefunction has; it is these coefficients that we need to find.

The number of plane-waves required to describe a real system accurately can be very large, often 100 times more than the number of electrons in the system. In other words, we have a very large matrix H, but only require the lowest few percent of its total eigenstates.

Our main aims today are:

- see how the time taken to solve eigenvalue problems scales with the size of the problem
- show how iterative diagonalisation methods can make large eigenvalue problems tractable

2. The Hamiltonian

The particular matrix we shall be concentrating on contains two of the main terms in a density functional theory-based materials modelling program,

$$H = T + V_{\text{local}}$$

where T is diagonal and represents the kinetic energy, and the V_{local} matrix is the electrostatic potential due to the nuclei.

The kinetic energy of each state is just half its wavevector squared, so if we order the plane-waves with the lowest wavevectors first, we can write

$$T_{nm} \propto n^2 \delta_{nm}$$

T is diagonal in G-space (Fourier space) so we only need to store the elements on the diagonal, but it is still a matrix. In contrast the nuclear potential matrix is diagonal in r-space (real space), so if we Fourier transform the wavefunction the nuclear potential matrix becomes diagonal and we only need to store those diagonal elements.

As in a true density functional theory calculation, the wavefunctions are complex-valued, and the Hamiltonian is Hermitian (taking the complex conjugate of the matrix is the same as its transpose). In general we have to solve a 3D problem, but in this Practical we will work with a simplified 1D version. In 1D there is one plane-wave with wavevector G=0, and then for every nonzero G there is one plane-wave with wavevector G and another with wavevector -G; i.e. the total number of plane-waves is (2*no. wavevectors + 1). As we increase the number of wavevectors in our calculation the number of plane-waves increases, and we allow our wavefunction(s) to have more Fourier components.

3. Exact Diagonalisation

Algorithms to find the eigenvectors of a matrix are well-known, and several are available in the linear algebra library LAPACK. The Hamiltonian matrix is always Hermitian which simplifies the problem.

Look at the skeleton code `eigensolver.f90`. You will see that this calls a number of subroutines that construct a large matrix from the various terms discussed above. There are several important variables that control the size of the problem:

- `num_wavevectors` is the number of nonzero Gs to include in the calculation, and determines the number of plane-waves num_pw (num_pw=2*num_wavevectors+1). Each eigenstate is a vector of length `num_pw` and the Hamiltonian is a `num_pw`×`num_pw matrix`.
- `num_states` is the number of eigenstates we wish to compute, and for most of this Practical we shall just set this to 1. We shall assume in this Practical that we will always be interested in the eigenstate(s) with the *lowest* eigenvalue (i.e. the most negative).

Using the LAPACK subroutine ZHEEV (or an equivalent), diagonalise the matrix H for

- `num_wavevectors=1`
  (which gives num_pw=2×1+1=3 so H is a 3x3 matrix)
- How does the time taken for the diagonalisation subroutine scale as you increase `num_pw`?
  Try increasing in steps up to a few thousand. You might like to fit a cubic polynomial to data.
- What happens to the lowest eigenvalue as you increase `num_wavevectors`? How many wavevectors are needed before the lowest eigenvalue is converged to two decimal places?
- Estimate how long it would take to diagonalise a 11,000 x 11,000 matrix (*but don't try it*).

NB since the LAPACK subroutine gives *all* of the eigenstates there is no need to worry about the value of `num_states`.

NB this code has to use a LAPACK library and the FFTW3 library (a particular FFT library). The Makefile already does this, so you should not need to do anything beyond type `make`.

4. Iterative Diagonalisation

We're now going to find the eigenstates using various iterative techniques. All of the iterative techniques work by starting from an approximation to an eigenvector, and improving it (somehow). For our problem we are going to exploit a property known as the Rayleigh-Ritz Variational Principle.

4.1 Steepest descents

If we have a guess for an eigenstate, ψ (stored in "trial_wvfn" in the skeleton programs), then we can compute a guess for the eigenvalue E as

$$E = \frac{\psi^\dagger H \psi}{\psi^\dagger \psi}$$

The gradient direction g is obtained by differentiating with respect to $\psi^\dagger$:

$$g = \frac{\delta E}{\delta \psi^\dagger} = \frac{(\psi^\dagger \psi) H\psi - (\psi^\dagger H x)\psi}{(\psi^\dagger \psi)^2}$$

If we assume the eigenstates are normalised then $\psi^\dagger \psi = 1$ and $E = \psi^\dagger H \psi$ so this simplifies to

$$\frac{\delta E}{\delta \psi^\dagger} = H\psi - E\psi$$

This tells us that if we add a bit of this vector to our guess, the estimated eigenvalue will increase quickly. We want it to *decrease*, so we just use minus this, i.e. the search direction

$$\phi$$

is:

$$\phi = E\psi - H\psi$$

and we perform a *line search* in this direction to find the new x vector,

$$\psi^{new} = \psi + a\phi$$

All we have to do now is keep adding more and more of this vector *search direction* until we find the smallest estimated eigenvalue in that direction. This gives us a better vector ψ, and we can go back and compute a new gradient direction. We do have to take care that at each step we normalise our trial eigenvector, since we have assumed normalisation in this derivation. The basic algorithm is:

1. Fill x with random numbers to start with
2. Apply the Hamiltonian to get $H\psi$
3. Compute the estimated eigenvalue $E = \psi^\dagger H\psi$
4. Form the gradient $H\psi - E\psi$ and the search direction $S = -(H\psi - E\psi)$
5. Search along S to find the new ψ vector
6. Go to step 2

This method is called the method of steepest descents, and each pass through steps 2-6 is one iteration of the algorithm. As we're written it above this algorithm will never stop - in practice we look at the change in the estimated eigenvalue as we go through each iteration, and when the change is less than a certain threshold we decide x is "close enough".

- Implement the steepest descents method using the framework in `eigensolver.f90`.
- How does it perform for `num_wavevectors=1`, `num_states=1`?
- What happens for `num_wavevectors=5`? What about 25? 50?
- The program writes the nuclear potential to the file "pot.dat" and the lowest wavefunction to "wvfn_1.dat". Plot these (e.g. in gnuplot or xmgrace) to see what they look like. What happens as `num_wavevectors` is increased?

4.2 Preconditioning

As `num_wavevectors` is increased, the kinetic energy part becomes increasingly dominant. The lower right-hand part of the Hamiltonian becomes diagonally dominant, and so the largest eigenvalue increases as the square of `num_wavevectors`. Unfortunately steepest descent schemes work best when all the eigenvalues are of the same magnitude, and consequently the algorithm takes longer and longer to converge as `num_wavevectors` is increased.

One solution to this problem is to multiply the search direction by something which will remove (or approximately remove) these unwanted terms - this is called *preconditioning*. If we write our approximate lowest eigenstate ψ in terms of the true eigenstates z_p, which have eigenvalues μ_p, we have

$$\psi = z_1 + \sum_{p\neq 1} \alpha_p z_p$$

We can see that the error is just

$$\sum_{p\neq 1} \alpha_p z_p \tag{1}$$

where, hopefully, the alphas are small. If we compare that to our gradient direction,

$$\begin{aligned} H\psi - E\psi &= (H-E)z_1 + \sum_{p\neq 1} \alpha_p (H-E) z_p \\ \Rightarrow \qquad S &\approx \sum_{p\neq 1} \alpha_p (H-E) z_p \end{aligned}$$

where we have assumed that E is close to λ_1. This is not the same as our error (equation 1), so it's impossible to go straight from our approximate eigenstate to the true one just by adding our gradient vector to it.

For the plane-waves with large wavevectors the kinetic energy term is dominant. If we assume the contributions from the other parts of the Hamiltonian cancel out, we can write

$$H - E \approx H_{kin} - E_{kin}$$

where we can compute E_{kin} for any particular state x as

$$E_{kin} = \psi^\dagger H_{kin} \psi = \sum_{p=1}^{numpw} \psi^\star(p) H_{kin}(p) \psi(p)$$

Because this matrix is diagonal, the inverse is also diagonal with elements

$$(H_{kin} - E_{kin})^{-1}$$

in fact since constant factors make no difference, it is common to use

$$\left(\frac{H_{kin}}{E_{kin}} - 1 \right)^{-1}$$

instead. This is called a kinetic energy preconditioner.

For the small wavevectors it is less obvious how to compute and invert (H-E). The safest thing is to leave these wavevectors alone, which means we need a function that will change smoothly from 1 at low wavevectors, to the kinetic energy preconditioner at high wavevectors. There are many such functions, but one particularly useful one is

$$f(x) = \frac{8 + 4x + 2x^2 + x^3}{8 + 4x + 2x^2 + x^3 + x^4} \tag{2}$$

where

$$x = \frac{H_{kin}}{E_{kin}}$$

To see why this is so useful, plot this function (e.g. in gnuplot or xmgrace) along with 1/(x-1). Remember that kinetic energy is always positive, so $x > 0$. Why is this better than 1/(x-1)?

The preconditioning matrix, P, is now given by:

$$P_{nm} = \delta_{nm} f(n)$$

(where f is the function from equation 2.)

- Implement this kinetic energy preconditioner in the subroutine "precondition". Remember to compute the kinetic energy eigenvalue for your approximate eigenstates.
- Use the precondition subroutine to create your search direction S from your gradient direction. Since the search direction has to be orthogonal to the approximate eigenstates you will need to re-orthogonalise after preconditioning.
- Compare the performance of the preconditioned and non-preconditioned steepest descents algorithm for the same parameters.

4.3 Conjugate Gradients

By keeping a history of previous search directions, we could build up a picture of what the Hamiltonian is actually like. This is the basis behind the conjugate gradient schemes. In the conjugate gradient (CG) method the search direction for the first iteration is (preconditioned) steepest descent, i.e. $S_1 = -\frac{\delta E}{\delta\psi^\dagger}$ or $S_1 = -P\frac{\delta E}{\delta\psi^\dagger}$ (where P is the preconditioner), but for subsequent iterations:

$$S = -\frac{\delta E}{\delta\psi^\dagger} + \gamma\phi_{prev}$$

In the Fletcher-Reeves CG variant we have

$$\gamma = \frac{g^\dagger P g}{g_{prev}^\dagger P g_{prev}}$$

where $g = \frac{\delta E}{\delta\psi^\dagger}$.

- Implement the conjugate gradient scheme of Fletcher-Reeves. Remember that your search direction y must always be orthogonal to the approximate eigenstates x.
- You will probably find the CG needs to be reset to steepest descents every so often to be efficient. Try resetting every 5, 10 or 20 steps.
- Compare the performance of this CG with that of the steepest descent algorithm.

The constraint that the gradient should be orthogonal to the approximate eigenstates can cause CG problems, which is why we sometimes have to restart it. We could also look at the change in eigenvalue and/or the magnitude of $S \dagger x$ to determine when to reset to steepest descents.

5. Advanced Work

If you've got this far you've done very well! If you still have the time you might like to try the following:

5.1 Multiple Eigenstates

So far we've only used the iterative methods to obtain the lowest eigenstate. Generalising the method to multiple states means:

- Instead of minimising the individual eigenvalues, we now minimise the energy, which is the sum of the eigenvalues we're interested in.
- The gradient and search directions must be orthogonal to *all* of the approximate eigenstates (what would happen if they weren't?)
- Run your eigensolver for `num_wavevectors=1` and `num_states=2`. What happens?

You may find you need to implement a new subroutine "diagonalise". This takes a linear combination of eigenstates and the Hamiltonian acting on those states, and rotates the states to be as close as possible to "true" eigenstates.

- Plot the lowest and second lowest wavefunctions in gnuplot ("wvfn_1.dat" and "wvfn_2.dat" respectively). How do they differ? Why?
- For `num_wavevectors=1500`, how many states do you need to compute before exact diagonalisation is more efficient than iterative diagonalisation?
- You should find your program is spending almost all of its time in orthogonalise, orthonormalise and transform - can you improve performance?
- How long does it take your program to solve the eigenvalue problem for `num_wavevectors=5,000`, `num_states=164`?