One-dimensional quantum harmonic oscillator

Abstract

In this work we write a Fortran program to compute the first k eigenvalues E_k and eigenvectors $|\psi_k\rangle$ of an one-dimensional quantum harmonic oscillator.

1 Theory

The Hamiltonian of the one-dimensional harmonic oscillator is:

$$\hat{H} = \frac{\hat{p}^2}{2m} + \frac{1}{2}k\hat{x}^2 = \frac{\hat{p}^2}{2m} + \frac{1}{2}m\omega^2\hat{x}^2, \tag{1}$$

where m is the particle's mass, k is the force constant, $\omega = \sqrt{k/m}$ is the angular frequency of the oscillator, $\hat{x}$ is the position operator and $\hat{p} = -i\hbar\frac{\partial}{\partial x}$ is the momentum operator. The first term in the Hamiltonian represents the kinetic energy of the particle, whereas the second term represents its potential energy.
One may write the time-independent Schrödinger equation,

$$\hat{H}\,|\psi\rangle = E\,|\psi\rangle \tag{2}$$

where E denotes a real number that will specify a time-independent energy level, or eigenvalue, and the solution $|\psi\rangle$ denotes that level's energy eigenstate. One may solve the differential equation representing this eigenvalue problem in the coordinate basis, for the wave function $\langle x|\psi\rangle = \psi(x)$, using a spectral method. It turns out that there is a family of solutions:

$$\psi_n(x) = \frac{1}{\sqrt{2^n\,n!}}\left(\frac{m\omega}{\pi\hbar}\right)^{1/4} e^{-\frac{m\omega x^2}{2\hbar}} H_n\left(\sqrt{\frac{m\omega}{\hbar}}x\right), \qquad n = 0, 1, 2, \dots \tag{3}$$

where the functions H_n are the physicist's Hermite polynomials,

$$H_n(z) = (-1)^n\; e^{z^2}\,\frac{d^n}{dz^n}\left(e^{-z^2}\right). \tag{4}$$

The corresponding energy levels are

$$E_n = \hbar\omega\left(n + \frac{1}{2}\right). \tag{5}$$

To numerically solve the Schrödinger equation we introduce two parameters: the system size L_s, which defines an interval $[-L_s, +L_s]$, and the spacing Δx. Given these two quantities, we will use the discretization $x_n = n\Delta x$ with $n = -L, -L+1 \dots, L-1, L$, such that $L_s = L\Delta x$. This leads to a total of $2L+1$ points.
In order to properly choose the parameters L, ω and Δx, we use the fact that the quantum harmonic oscillator possesses natural scales for length and energy. In particular, the typical length scale is given by

$$x_0 = \sqrt{\frac{\hbar}{m\omega}}, \tag{6}$$

so we need to choose L and Δx such that $2L_s \gg x_0$.

2 Code development

In order to compute the Hamiltonian, compute its eigenvalues/eigenvectors and find the probability density corresponding to the eigenvectors, we define a module `HarmonicOscillator1D` (contained in `harm_osc_1D.f90`), which contains the following subroutines:

- `DiscretizedLaplacian`, in which we compute the discretized laplacian. To do that, we use the fact that the second derivative of $\psi(x_n) \equiv \psi_n$ can be discretized as

$$\psi_n'' \approx \frac{\psi'_{n+1} + \psi'_{n-1} - 2\psi'_n}{(\Delta x)^2}$$

  which can be rewritten in a matrix form as

$$\psi'' \approx \frac{1}{(\Delta x)^2} \begin{pmatrix} -2 & 1 & 0 & \cdots & 0 \\ 1 & -2 & 1 & \ddots & \vdots \\ 0 & 1 & -2 & \ddots & 0 \\ \vdots & \ddots & \ddots & \ddots & 1 \\ 0 & \cdots & 0 & 1 & -2 \end{pmatrix} \psi;$$

- `HarmonicPotential`, in which we compute the potential for the harmonic oscillator, which can be written as a diagonal matrix with elements $\frac{1}{2}m\omega^2 x_n^2$;
- `ComputeEigenvalues`, in which we compute the eigenvalues and the eigenvectors of the Hamiltonian using the LAPACK `zheev` subroutine (which is analogous to `cheev`, but it works with `complex*16` arrays). In this case we call this subroutine two times: the first to find the optimal parameter `lwork`, the second to do the actual computation;

```
    ! ------------ find the optimal lwork -----------------------------
    lwork = -1
    allocate(work(1))
    call zheev(jobz, uplo, n, matr, lda, eig, work, lwork, rwork, info)
    ! on exit, if info = 0, work(1) returns the optimal lwork
    lwork = int(work(1))
    deallocate(work)
    ! -----------------------------------------------------------------
    ! allocate work using the optimal lwork
    allocate(work(lwork))
    ! perform the diagonalization
    call zheev(jobz, uplo, n, matr, lda, eig, work, lwork, rwork, info)
```

- `ComputeProbabilityDens`, which is a simple subroutine that squares the absolute value of the eigenvectors to find the corresponding probability density;
- `ComputeThEnergy`, which computes the theoretical eigenvalues.

Then, we define another module, `Utilities` (contained in `util.f90`), which contains some useful tools for real/integer-to-string conversion and subroutines to save the results in output files.

In the main program (`harmonic_oscillator_1D`) we ask the user if he/she wants to enter custom values for $L, \Delta x, \omega, m$ and $\hbar$ or if he/she wants to use default ones. Then we compute the total number of points $N = 2L + 1$ and we construct out Hamiltonian using the subroutines seen above.

```
    H = -((hbar**2)/(2*m*dx**2))*laplacian + harmonic_potential
```

Successively, we diagonalize it and we find the probability densities. Furthermore, we also compute the differences between numerical eigenvalues and theoretical ones.
Finally, we save the results in different text files which are named according to the chosen parameters. The notation is the following: `data_L_Dx_omega_m_hbar.txt`, where `data` refers to the types of data (`en` = energies, `ef` = eigenfunctions, `pr` = probability densities, `en_err` = energy errors). The plots are done using a Python script that runs different gnuplot files.

We also want to compare the numerical eigenvectors with theory. To do that, we use the module `hermite` (contained in `th_eigenf.90`) which builds the Hermite polynomials that are needed to compute the eigenfunctions Eq. (3). Again, the results are saved in different text files which are named according to the chosen parameters. The notation is the following: `data_t_k_L_Dx_omega_m_hbar.txt`, where `t` stands for "theoretical" and `k` is the number of eigenvalues we want to consider.

3 Results

As mentioned before, in order to get meaningful results we need to properly choose the parameters L, Δx and ω. After some trials, the following conclusion is drawn: if we fix ω and L, by considering finer and finer spacings Δx we get worse results, in particular for the eigenvalues. This is because by taking smaller and smaller Δx we are not able to see the quadratic potential, which will look more and more like an infinite well. On the other hand, if we take Δx too big ($\Delta x \sim 0.1$) we would get again worst results, especially for the eigenvectors. The errors in the computation of the eigenvalues for different Δx – computed by simply taking the difference between numerical and theoretical results – are shown in Figure 1. As we can see, the best value for Δx seems to be of the order 10^{-2}.

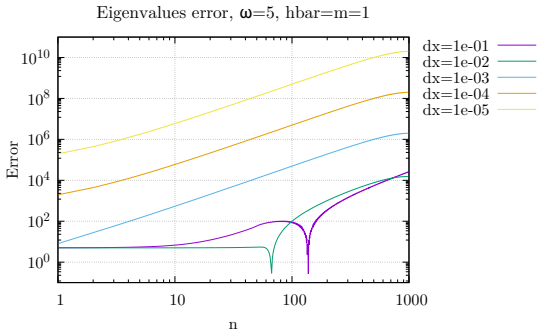


Figure 1: Eigenvalues computational error for different Δx.

Consider, as a quite good example, the case $L = 500$, $\Delta x = 0.01$ and $\omega = 5$. This way we have $2L_s = 10 \gg x_0 \approx 0.4$.
The eigenvalues found using this parameters are shown in Figure 2. We can see that the numerical results do not agree so much with the theoretical prevision. This is certainly due to the discretization we introduce, that forces us to set some boundary conditions. In our case we set the eigenvectors to be zero at the bounds, which – acting like an infinite potential well – has some effect on the eigenvalues. However, as we can see in the inset plot, the approximately first 70 vales are coherent, which is a good result (in other cases not even the first 10 are correct).

The first $k = 5$ eigenfunctions are shown in Figure 3 and the corresponding probability densities are shown in Figure 4. Notice that some of the eigenfunctions are flipped: this can

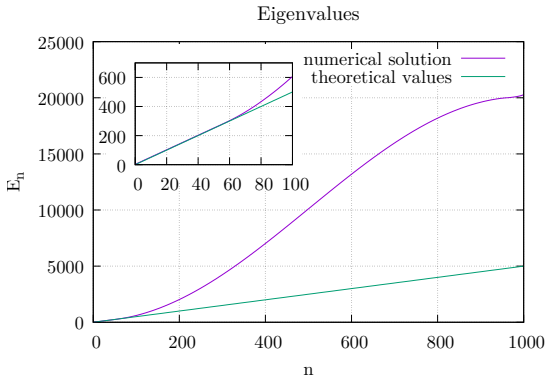
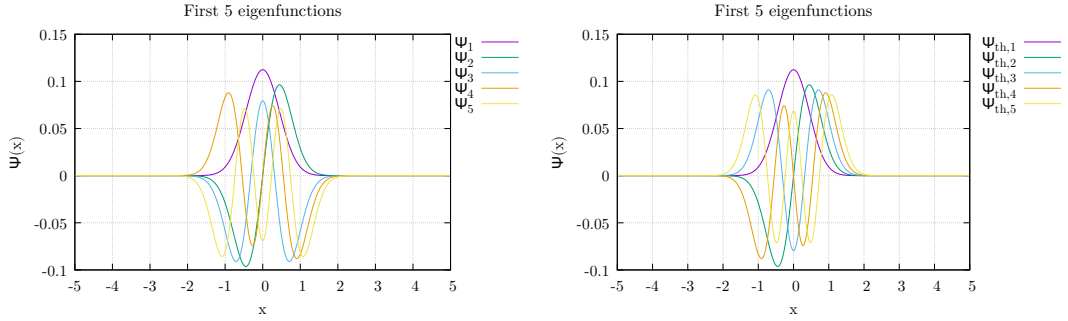


Figure 2: Eigenvalues for $\hbar = m = 1, L = 500, \Delta x = 0.01$ and $\omega = 5$.



(a) Numerical results.

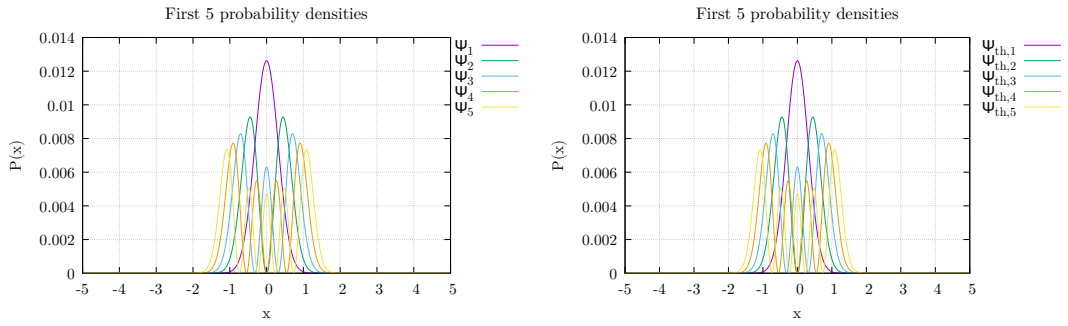
(b) Theoretical results.

Figure 3: First $k = 5$ eigenfunctions for $\hbar = 1$, $m = 1$, $L = 500$, $\Delta x = 0.01$ and $\omega = 5$.

happen, since we know that the eigenfunctions are defined up to a multiplicative constant -1. In fact, if we look at the probability densities, they all agree.

3.1 Program rating

- Correctness. While writing the code, the module `debugger` – which can be found in the `debugger.f90` file – was widely used to perform incremental testing. Moreover, the code was validated using a wide range of input paramters.
- Stability. The code was written trying to avoid floating point arithmetic errors, for example comparisons/repeated sums of floating point variables or catastrophic cancellation.
- Accurate discretization. This question is discussed in the Results section. Essentially, after various tests led to a discretization of the order $\Delta x = 10^{-2}$, which leads to good results both for the eigenfunctions and the eigenvalues (the first 70 are coherent with theory. If instead we choose $\Delta x = 10^{-3}$ only the first two or three are correct).
- Flexibility. The code allows to choose all the possible parameters, so it is quite flexible. Moreover, if one wants to consider different types of potential, they could be easily added to the module as subroutines.



(a) Numerical results.

(b) Theoretical results

Figure 4: First $k = 5$ probability densities for $\hbar = 1$, $m = 1$, $L = 500$, $\Delta x = 0.01$ and $\omega = 5$.

- Efficiency. If we consider the program itself, it is very basic and the the most expensive computation is the diagonalization of the Hamiltonian. This is done by means of the subroutine `zheev`, which was optimized by computing the best `lwork` parameter. This way, we get for sure a better efficiency with respect to a user-defined subroutine. If we then consider that the parameters have to be properly chosen, from this point of view the program is not so efficient because the tuning is done by hand executing the program many times with different parameters. One possible improvement could be, for example, to implement a grid search.

4 Self evaluation

This exercise was very instructive since it made me learn how to numerically solve a problem that usually is only "theoretically" solved. It was very interesting.

One possible improvement could be to better organize the saving and plotting of the results. In other words, one could try to better automate the process of plotting from different data files. This is quite challenging, since the Fortran program's output files will have a different name every time the code is run with different parameters. At the moment, the user has to manually change the name of the file to plot inside the gnu files. One could for example add to the Fortran code an additional output file `filenames.txt` which is open with `status="old"` and in which the name of the output files is written every time the code is run. Then one should modify the Python script to read the data file names from `filenames.txt` and automatically do the plots.

P.S. In virtue of the motto "less is more" I could have plotted both numerical and theoretical results in the same figure. However, even choosing different line styles, the result was not so readable, so I preferred to make two separate plots.