Quantum information and computation: homework 6

of Tommaso Tabarelli

Abstract

In this homework we are asked to analyze the results of the analysis of the *unidimensional quantum harmonic oscillator*. In particular we are asked to compare computer eigenvalues and eigenvector results with theoresical ones. We should produce these results using a Fortran program.
To do it we should write the discretize matrix form of the *Hamiltonian* operator $\hat{H}$ act on this representation using *DSYEV* lapack routine, in particular we should compute eigenvalues and eigenvectors.

Theory

The unidimensional quantum harmonic oscillator is the quantum version of the classic harmonic oscillator in which a particle of mass m is attached to an extreme of an ideal spring, this one having the other extreme fixed in the origin.
Naming x the spring extension (with respect to the origin), p the particle linear momentum and k the spring constant, the classical system is described by the following equation for the energy:

$$H = \frac{p^2}{2m} + \frac{1}{2}kx^2$$

In the quantum case one should use the corresponding operators:

$$\hat{H} = \frac{\hat{P}^2}{2m} + \frac{1}{2}k\hat{X}^2$$

and to find the energies, the eigenvalues problem to solve is:

$$\hat{H}|\psi_E\rangle = E|\psi_E\rangle$$

The problem now is that we want to write $\hat{H}$ in a matrix form to analyze it. To do it a representation form has to be chosen. We shall use the x-representation for both $|\psi\rangle$ and $\hat{H}$. In this representation we have:

$$\hat{P} = -i\hbar\frac{\partial}{\partial x} \qquad \hat{X} = x$$

So the $\hat{H}$ eigenvalues equation in this representation becomes (using also $\omega = \sqrt{k/m}$):

$$-\frac{\hbar^2}{2m}\frac{\partial^2}{\partial x^2}\psi_E(x) + \frac{1}{2}m\omega^2x^2\psi_E(x) = E\psi_E(x)$$

It can be shown that the eigenvalues are:

$$E_n = \hbar\omega\left(n + \frac{1}{2}\right) \quad n = 0, 1, 2, ...$$

and the eigenfunctions are (in x-representation):

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

where $H_n(\xi)$ are the *Hermite polynomials* of order n.

Since in a computer continuos variable can not be represented, to write down the matrix form of the matrix one has to fix a discretization. The only thing to discretize is the second order derivative, which has the form (naming h the discretization spacing):

$$\frac{\partial^2}{\partial x^2} f(x) \simeq \frac{f(x-h) - 2f(x) + f(x+h))}{h^2}$$

Naming x_i the points of the interval, we have that $x_i \pm h = x_{i\pm1}$. Thus the discretized Hamiltonian has the form:

$$H = \begin{pmatrix} \frac{2\hbar^2}{2mh^2} + \frac{1}{2}m\omega^2x^2 & -\frac{\hbar^2}{2mh^2} & 0 & \dots & & 0 \\ -\frac{\hbar^2}{2mh^2} & \frac{2\hbar^2}{2mh^2} + \frac{1}{2}m\omega^2x^2 & -\frac{\hbar^2}{2mh^2} & 0 & \dots & \vdots \\ 0 & -\frac{\hbar^2}{2mh^2} & \frac{2\hbar^2}{2mh^2} + \frac{1}{2}m\omega^2x^2 & -\frac{\hbar^2}{2mh^2} & 0 & \dots \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots \\ 0 & & \dots & & 0 & -\frac{\hbar^2}{2mh^2} \quad \frac{2\hbar^2}{2mh^2} + \frac{1}{2}m\omega^2x^2 \end{pmatrix}$$

Code development

The Fortran program first reads 2 files in which the size of the interval and the number of intervals to use for the discretization are written. Then variables are allocated. After that, the proper evaluation takes place: basing on the number of points, a proper Hamiltonian is created and it is used to feed lapack's *DSYEV* subroutine after having let it found the optimal dimension to work with. Increasing order eigenvalues are collected and stored. The Hamiltonian is changed: after the subroutine call it contains the "normalized" eigenvectors. It is important to notice that these eigenvectors are not properly normalized: by construction they represent the eigenfunctions evaluated in the points of the considered interval. *DSYEV* normalizes the norm of them, thus it does:

$$\|\phi(x)\| = 1 \quad \Rightarrow \quad \sqrt{\sum_i \phi(x_i)^2} = 1 \quad \Rightarrow \quad \sum_i (\phi(x_i) \cdot 1)^2 = 1$$

where it is emphasized the fact that this normalization is similar to ask that the integral of the squared function is 1 using all *size-1* intervals.
What we want is instead:

$$\int_{-\infty}^{+\infty} \|\psi\|^2 dx = 1 \quad \overset{\text{discretizing}}{\Rightarrow} \quad \sum_i \left(\psi(x_i)^2 \cdot h_i\right) = 1 \quad \Rightarrow \quad \sum_i \left(\psi(x_i) \cdot \sqrt{h_i}\right)^2 = 1$$

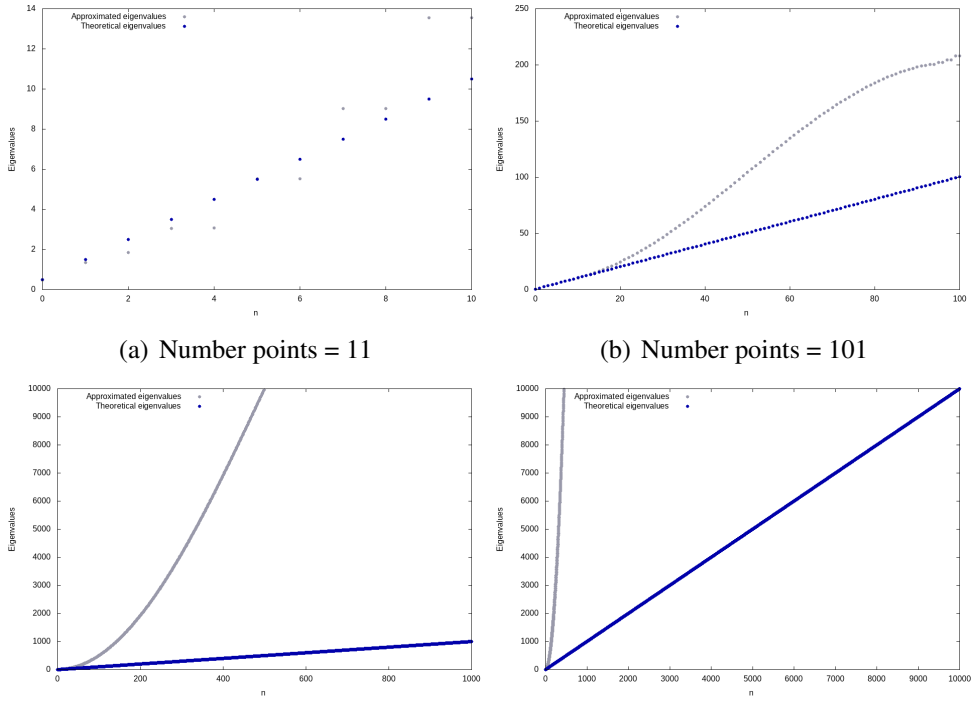
It is immediate now to see that to get the properly normalized eigenfunction one has to divide the *DSYEV* eigenvectors by $\sqrt{h}$ where h is the common spacing, i.e. the size of the discretization interval:

$$\psi(x_i) = \frac{\phi(x_i)}{\sqrt{h}}$$

Theoretical eigenfunctions are also evaluated: to do it, 2 modules are defined. The first is used to evaluate Hermite polynomials while the second is used to evaluate the factorial value of a number. All results are stored to proper files.

To execute a better task and explore different parameters values a python script was implement, acting on different interval sizes and number of points. This script also creates proper folders in which store the data, saving current date and time as folder name.
The python script also takes care to plot eigenvalues results and compare the first 10 (arbitrary value choice to try to limitate memory issues) approximated and theoretical eigenfunctions.

Results



(a) Number points = 11

(b) Number points = 101

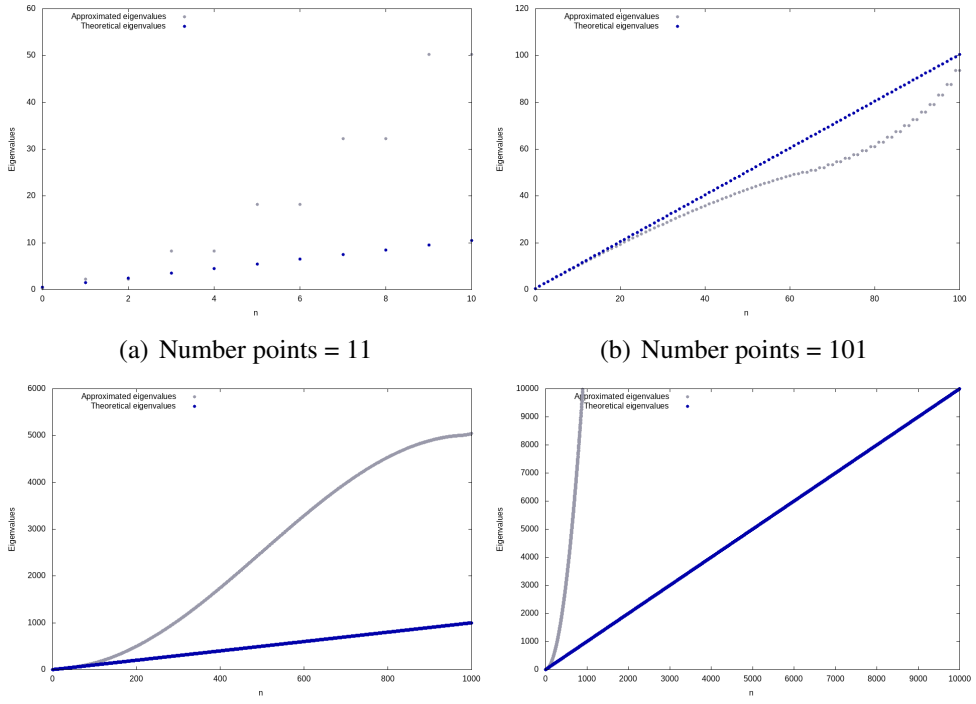
(c) Number points = 1001

(d) Number points = 10001

Figure 1: Eigenvalues evaluated using the interval [-5,5]

The images show that the discretization plays an important role in determining the eigenvalues approximations. Indeed increasing the number of points in the interval leads to a better discretization untill a certain threshold, which is around 100; after that, the first eigenvalues are still approximated correctly, but the successive ones have a divergent behaviour and can not be considered correct (the eigenvalues are not even written to the file due to their dimension not being compatible with the Fortran format chosen; this has been fixed but the program has not been run again due to time issues). The divergent behaviour is due to the fact that a finite interval is used to evaluate the approximations: the (implicit) boundaries conditions are that the function ψ is 0 out of the interval, causing the potential to become an infinite well and the eigenvalues to follow the corresponding quadratic behaviour (which starts after a certain point) and to have degeneration factor equal to 2 (as can be noticed well in the Figure 2a and 2b).

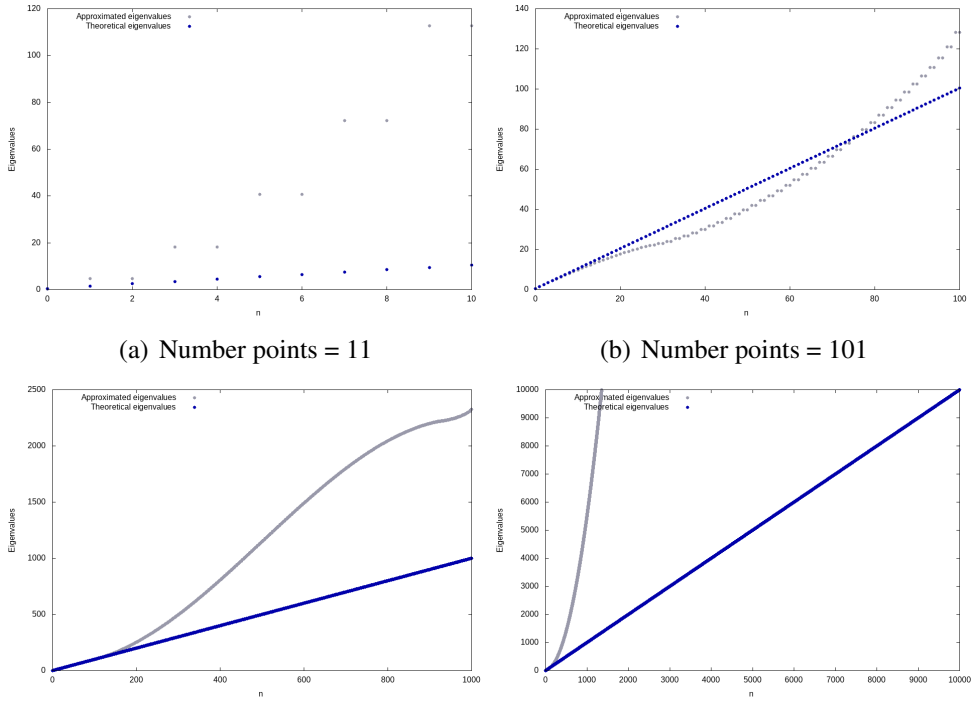
Now, to see if the first eigenvalues are "correct" we shall see if the eigenvectors match the theoretical ones. To do this, the choice was to compare the first ten eigenvectors (manipulated as explained above) and the first ten theoretical eigenfuntcions. The reported results are those of the $n = 9$ eigenvector for L=5 and different k (Figure 4).



(a) Number points = 11 (b) Number points = 101

(c) Number points = 1001 (d) Number points = 10001

Figure 2: Eigenvalues evaluated using the interval [-10,10]

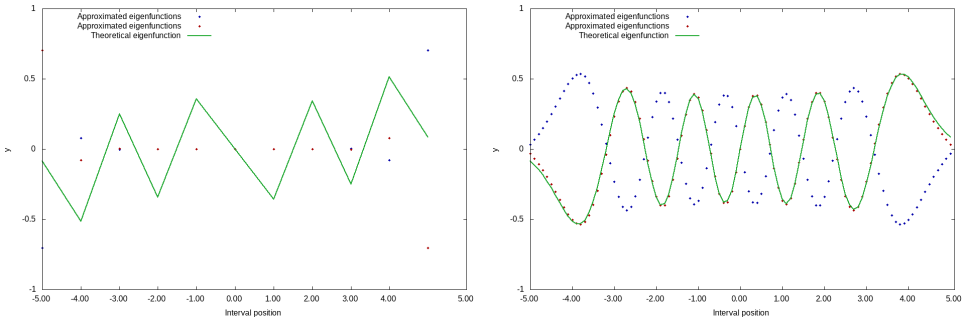


(a) Number points = 11 (b) Number points = 101

(c) Number points = 1001 (d) Number points = 10001

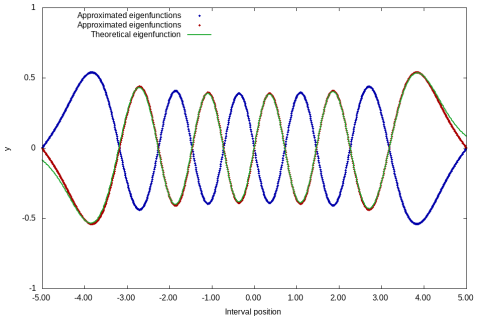
Figure 3: Eigenvalues evaluated using the interval [-15,15]

As one can notice, the *boundaries effect* (in the Hamiltonian discretization) also affect the function evaluation over the points outside the interval, which are considered 0 (indeed they do not appear in the discretization). This leads to a bad approximation near the interval extrema. To try to fix it one

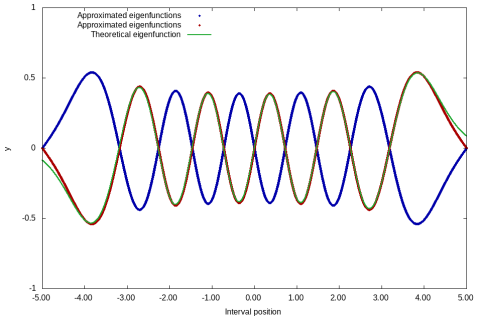


(a) Eigenfuntcion: n = 9

(b) Eigenfuntcion: n = 9



(c) Eigenfuntcion: n = 9



(d) Eigenfuntcion: n = 9

Figure 4: Eigenvectors evaluated using the interval [-5,5] using 10 points.

can chose a larger interval; using the L=15 indeed leads to better results (but of course this does not fix the eigenvalues problem aforementioned).

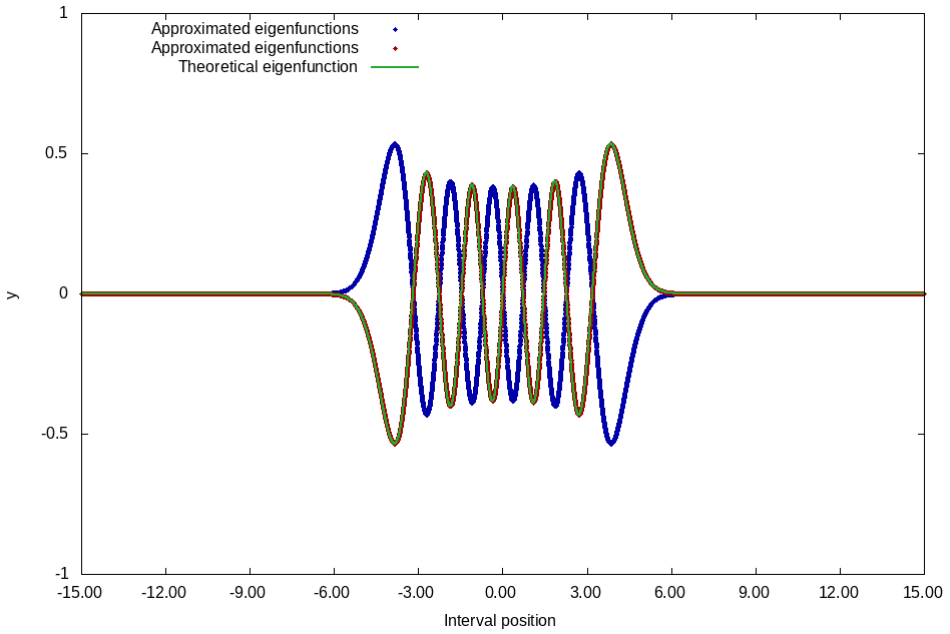


Figure 5: Eigenvectors with n=9 evaluated using the interval [-15,15] using 10000 points.

Self-evaluation

In this task I learned how to discretize and properly write in a program the Hamiltonian operator in the x-representation. I also learned how to get approximated eigenvectors and how manipulate them in a proper way to compare them with theoretical ones. Another important thing was to learn how to write down properly normalized *harmonic oscillator* eigenfuntcions; to do it I had to implement a *factorial* subroutine and a *Hermite polynomial* subroutine.
For what concerns *Correctness, Stability, Accurate discretization, Flexibility, Efficiency* the considerations are the following.

Correctness

My program does what is expected to do. Some checks are made both in Python script and in Fortran program (for example dimensions are tested to be greater than 0).
There are no compilation problems neither memory allocation issues (a special flag is used to compile in order to check for memory error runtime).
Pre- and post-conditions and checkpoints are used for the matrix H, but they are implemented manually and are not used very much in the program due to the large amount of time they require to be implemented (compared to the time we have to do the task).

Stability

In the program *IMPLICIT NONE* is used to avoid strange behaviours due to the implicit type of the variables.
Loops are made only on integers (usually the extreme values of the loop variables are the same used to initialize the dimension of the objects).
Differences between big numbers are not evaluated in the program to avoid the problem of having finite precision.

Accurate discretization

As mentioned before, a python script was used to explore different interval width. Using too large intervals the results are bad (as one can see in the 11 points approximations), while using too many points rises memory and time problems (evaluations for 10001 points lasted about 30 min each and eigenvectors file size were about 1.3 GB each).

Flexibility

I tried to comment as much as possible the code so to have a clear and understandable code.

Efficiency

I did not implemented the diagonalization of the matrix; instead *DSYEV* lapack subroutine was used. To evaluate factorial and Hermite polynomials two subroutines were written, but they are quite simple and I think there can be slightly improvements to them.
For what concerns memory and time usage, they depend on the parameters passed. Also, time scales with input size (i.e. dimensions of the matrix) as $O(n^3)$, so for very high dimension it can take a lot

of time to end the task.

My work can be improved in different ways: first of all, other parameters can be taken as input in Fortran program (for example, mass and ω); then, to compare eigenfunctions it would be good looking for the eigenfunctions of "uncorrect" eigenvalues, such as those starting from the half of the eigenvalues list (say, starting from k/2 index).
Other improvements in the data analysis would be to evaluate the infitite well potential eigenvalues and eigenfunctions and compare them with the theoretical ones.