Numerical solution for the Quantum Harmonic Oscillator

Salvador Alberto
Università degli Studi di Padova
Course: Quantum Information and Computing
Assignment 4

November 26, 2024

Contents

1 Introduction

The Quantum Harmonic Oscillator (QHO) is one of the most fundamental systems in quantum mechanics. It serves as a model for various physical systems such as molecules, atoms, and even fields in quantum field theory. In this report, it is provided a computational study of this physical system, with a comparison between the numerical solutions and the analytical expressions for its energy levels and eigenfunctions.
In Sec.2 I provide some basic knowledge about the theory underneath the QHO, especially regarding the eigenvalue problem. In Sec.3 I present the tools and main ideas I have used to obtain a computational simulation of the QHO. In particular, in Sec.3.1 I describe the *Finite difference method* for obtaining a numerical approximation for the eigenvalues and eigenfunctions. In 3.2 I describe what are *sparse matrices* and why it is useful to employ them in this context. Finally in Sec.3.3 I discuss the main guidelines I have followed in order to develop reliable and robust code for implementing this simulation. In Sec.4 I discuss the results obtained.

2 Theoretical background

2.1 Theory of the Quantum Harmonic Oscillator

The Hamiltonian for the one-dimensional quantum harmonic oscillator is:

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

where $\hat{p}$ is the momentum, m is the mass of the particle, ω is the angular frequency, and $\hat{x}$ is the position. In the space-representation we have $\hat{p} = -i\hbar\frac{\partial}{\partial x}$. This Hamiltonian leads to the following time-independent Schrödinger equation:

$$\hat{H}|\psi_n(x)\rangle = E_n|\psi(x)_n\rangle \tag{2}$$

where $|\psi_n(x)\rangle$ and E_n are respectively the eigenfunction and eigenvalue associated to the quantized number n. Its solutions are well-known and can be found in every Introduction to Quantum Mechanics textbook, as [1]. We have:

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

and

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

where $H_n(x)$ is the *Hermite polynomial* of nth-order, i.e., a particular function in this famous class of orthogonal polynomials. Note: these functions are all real, having zero imaginary parts.
In the following, I adopt *natural units*, in particular: $\hbar = 1$, $m = 1$, $i = 1$. In addition, in all the calculations I set $\omega = 1$.

3 Numerical Methods

To solve the Schrödinger equation numerically, we need to discretize the domain on a grid of points:

$$x \in \mathbb{R} \quad \longrightarrow \quad x \in \{x_0, x_1, x_2, ..., x_N\}.$$

This operation results in eigenfunctions in the form of vectors of length N and in a Hamiltonian in the form of a $N \times N$ matrix. Their entries come from the evaluation of the analytical functions (3) on the grid's points. The diagonalization of such matrix returns the solutions to the Schrödinger equation (2). The evaluation of the Hamiltonian (1) on this grid is, however, not trivial: the gradient operator coming from $\hat{p}$ should be properly discretized. This can be done up to a certain order of precision, with the *finite difference method*.

3.1 Finite Difference Method

The time-independent Hamiltonian of the harmonic oscillator (1) can be written in natural units as:

$$\frac{1}{2}\left(-\frac{\partial^2}{\partial x^2} + \omega^2 x^2\right) = \hat{K} + \hat{V} \tag{4}$$

separating the kinetic term $\hat{K}$ from the potential one $\hat{V}$. The latter is a diagonal operator:$\hat{V} = \frac{\omega^2}{2}\text{diag}(x_1^2, x_2^2, \cdots, x_N^2)$. The matrix representation of $\hat{K}$ can be obtained in an approximated form using the finite difference method. This method considers a truncated Taylor expansion of the wave function and retrieves the second derivative from that. Let us consider the simplest case of a second-order expansion. Expanding the wave function $\psi(x)$ to the second order around $x = 0$, one obtains:

$$\frac{d^2\psi(x)}{dx^2} \approx \frac{\psi(x+h) - 2\psi(x) + \psi(x-h)}{h^2}$$

where h is the spacing between grid points and the function is evaluated in three points. This leads the following tri-diagonal matrix

$$\hat{K} = \frac{1}{2h^2}\begin{pmatrix} 2 & -1 & 0 & \cdots & \cdots & \cdots & 0 \\ -1 & 2 & -1 & 0 & \cdots & \cdots & 0 \\ 0 & -1 & 2 & -1 & 0 & \cdots & 0 \\ \vdots & 0 & \ddots & \ddots & \ddots & \ddots & 0 \\ \vdots & \vdots & \ddots & \ddots & \ddots & -1 & 0 \\ 0 & 0 & \cdots & 0 & -1 & 2 & -1 \\ 0 & 0 & \cdots & 0 & 0 & -1 & 2 \end{pmatrix} \tag{5}$$

From (4) we then obtain a tri-diagonal hamiltonian given by $\hat{H} = \hat{K} + \hat{V}$.
Further orders in the expansion of $\psi(x)$ can be considered, resulting in different expressions for the matrix $\hat{K}$. As an example, the third-order expansion gives rise to a penta-diagonal symmetric matrix with entries: $\frac{5}{2}$ for each element in the main diagonal; -$\frac{4}{3}$ for each element in the diagonals with offsets $k = \pm 1$ to the main one; $\frac{1}{12}$ for each element in the diagonals with offsets $k = \pm 2$ to the main one.

3.2 Sparse matrices

As can be easily seen from 5, a lot of the entries of $\hat{K}$ are zeros. This extends obviously also to $\hat{H}$. We define matrices like that *sparse matrices*. For them, the computational burden can be greatly optimized by storing only the non-zero elements: this leads to reduced memory usage and a lower computational cost associated with matrix operations. The gain increases with the size of the matrix, which, in this case, depends on the granularity of the discretized domain. In short: by exploiting the advantages of sparse matrices, we obtain significant improvements in precision when computing the solutions to the Schrödinger equation. Note: as the order of approximation in finite difference methods increases, the number of non-zero diagonals grows, potentially diminishing the advantages of sparse matrix methods.
In Python, the `scipy.sparse` library provides a comprehensive suite of tools for dealing with sparse matrices. In my code, I have left optional the choice of working with sparse matrices or "dense" ones, with a boolean flag that selects which of the two to use. After effectively noticing the advantages in terms of computational time of sparse matrices, I decided to employ exclusively them througout my computations.

3.3 What we define as 'good' code

One of the goals of this report is to be conscious of the quality of the code developed for solving the problem of the Quantum Harmonic Oscillator. Following [2], the key priorities every scientific developer should follow are:

- *Correctness*
  The software should execute instructions correctly and use proper syntax for the programming language used. To ensure correctness, employ checkpoints, "error and exceptions" code sections, and tests with various input types.

- *Numerical Stability* Errors due to differences between real numbers and their approximation with floating-point arithmetic must be avoided and (especially) prevented.

- *Accurate discretization* Numerical operations on continuous variables always involve some kind of discretization. This introduce an error that must be controlled: the simulation should be consistent with the physical reality. If this does not hold, a higher-order discretization must be taken into consideration.

- *Flexibility* Good software should be designed for future use and adaptability to changing requirements. Developers should think broadly during programming and allow for future developments. A useful approach is to develop modular code, where each module contains functions and objects related to the same application. Accurate comments and meaningful names for variables and functions also enhance flexibility, making the code easier to interpret and reuse for other applications.

- *Efficiency* At last, the algorithmic complexity of the code must be analyzed, and, if possible, optimized. One way to accomplish this is to use professional available libraries. Of course, this requires a basic knowledge of the methods they employ.

4 Results

4.1 Numerically computed eigenvalues and eigenfunctions

In the following, I discuss the code implementation I use to solve the eigenvalue problem for the Quantum Harmonic Oscillator (2). The programming language I use is Python.
Inside the module `matrix_method.py` I define a function to generate a generic multi-diagonal simmetric matrix, named `gen_diag_simmetric_mat`. The returned matrix can optionally be a `scipy.sparse.dia_matrix`. I also define a function, named `diagonalize_hermitian`, for diagonalizing such matrix and returning the first k eigenvalues (sorted in ascending order from the algebraically smallest). Inside of it, I use the optimized library for diagonalization of Hermitian matrices `scipy.sparse.linalg.eigsh`. `quantum_harmonic_oscillator_methods.py` is another module I define, which contains functions dedicated to the quantum harmonic oscillator simulation. In particular, I have the `QHO_diagonalization` function, which generate and diagonalize the hamiltonian discussed in Sec.3.1. I focus on its approximation to the second-order as it give satisfactory results. From it I obtain the first 10 eigenvalues and eigenfunctions of the QHO. The discretization I choose for the one-dimensional space domain considers a range from -10 to +10 with 1000 points, as this provide results well-according to the theoretical ones. Theoretical results are computed using (3) inside another function named `stationary_states`. Both numerical and theoretical eigenfunctions are normalized. In Fig.1I report an example of the result obtained for the eigenfunction of energy level $n = 2$. I plot the probability density $P(x) = |\psi_n(x)|^2$ to avoid mismatches between the two functions due to global phases. In Tab.1 I report the results for the eigenvalues estimates.

n	0	1	2	3	4	5	6	7	8	9
Theoretical E_n	0.500	1.500	2.500	3.500	4.500	5.500	6.500	7.500	8.500	9.500
Numerical E_n	0.5005	1.501	2.502	3.503	4.504	5.505	6.505	7.506	8.507	9.507

Table 1: Comparison of theoretical and numerical energy levels for the first 10 eigenvalues.

4.2 Results dependance on the discretization

Following the discussion from Sec.3.3, one of the key features for developing a good simulation of a physical system is how its continuous variables are discretized. In this case, we are concerned about the one-dimensional space discretization. To evaluate the impact of discretization on my results, I use two measures:

- *Energy error* The absolute difference betweeen the numerically computed eigenvalue and its theoretical counterpart.
- *Overlap* The scalar product between the numerically computed eigenfunction and its theoretical counterpart.

I consider a grid of parameters for the interval size, i.e. the distance between the initial point a and the final one b, and the number of points N in which it is discretized.

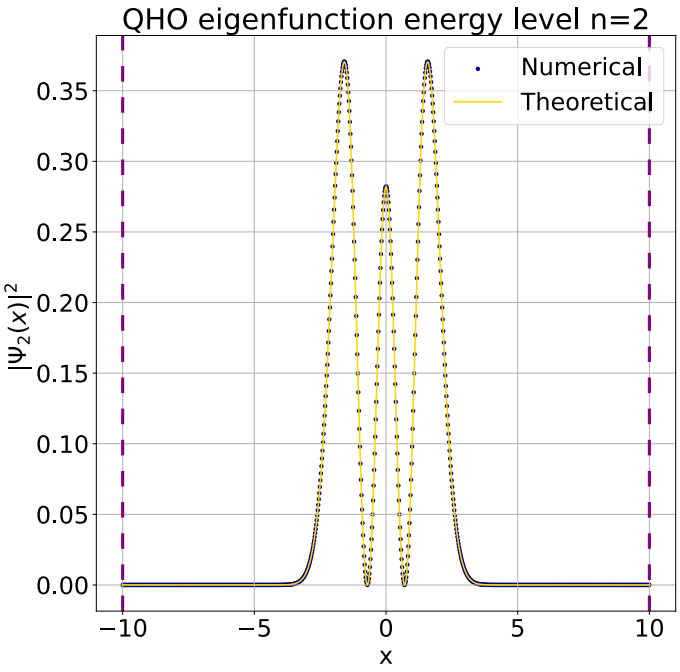


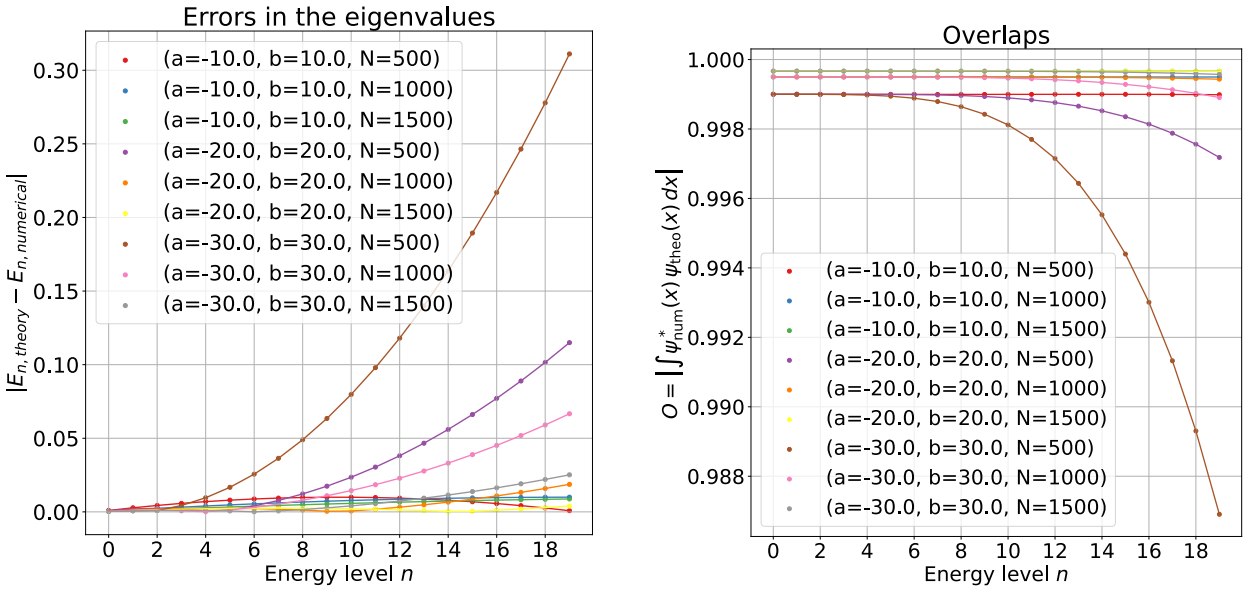
Figure 1: Wavefunction for the second energy level of the quantum harmonic oscillator. In the plot it is plotted both the analytical solution, obtained using (3), and the numerical one computed on a grid interval with boundaries -10 and 10 and a number of points equal to 1000.

In particular, I consider values in $[20, 40, 60]$ for the sizes and $[500, 1000, 1500]$ for N. I extend the number of eigenvalues and eigenfunction to the first 20. A specific function inside the `quantum_harmonic_oscillator_methods.py` module, called `QHO_errors_analysis`, completely provide this analysis. In Fig.2a and Fig.2b I present the results I obtain.

The evident outcome is that not all discretizations lead to the same results. One can see how having fewer points leads to bigger errors (brown and purple curves). Another factor that leads to increased errors is that the energy level considered: almost all the curves have an uprising (for the energy differences) and downfall (for the overlaps) trend for an increase in the energy level n. Summarizing: as you try to compute higher levels eigenvalues/eigenfunctions, you must consider a wider domain and an increased number of points.

4.3 Evaluation of my code

Based on the priorities that a good developer should follow from Sec.3.3, I now evaluate my code. In terms of *correctness*, the code works as expected, and I have not encountered any impediments while using it to obtain my results. I inserted various checkpoints to be shown in debugging mode and a boolean flag turn it on. I also tried to prevent errors, for example checking to have in input the correct type for subsequent use. Both the checkpoint and error subroutine are contained in the `debugger_module.py` module. In Sec.4.1 I discussed the type of modularity I developed for my code, which ensures *flexibility* for further usage of



(a) (b)

Figure 2: Comparison between the errors in the eigenvalues (a) and overlaps (b) for the numerically computed eigenfunctions and their theoretical counterpart. Each curve represents a different choice for the discretization of the one-dimensional space.

the code in possibly different contexts. Throughout the developement I thought in terms of what could be useful for future assignments, for instance, by defining ad-hoc functions for the generation of multi-diagonal matrices and their diagonalization, rather than focusing solely on the specific task of the QHO. Regarding *Accurate discretization*, in Sec.4.2 I studied the limits of my simulation in terms of results accuracy, related to the implemented discretization. I showed that a good choice for computing the first 10 eigenvalues and eigenfunctions can be using an interval of size 20 (from -10 to 10) and 1000 points. In terms of *numerical stability*, I designed the code to minimize the probability of making mistakes because of errors in number precision. Nevertheless, this type of error is very difficult to foresee. To ensure nothing inconvenient happens, one way is to make tests with different inputs and verify whether the results are consistent with expectations. Based on all the tests I performed, nothing unusal happened. Finally, in terms of *efficiency*, the run-time is sufficiently satisfactory, taking only a few seconds on my laptop. It could be interesting to consider more eigenvalues to determine the computational limit determined by the use of sparse matrices. However, for the purposes of this assignment, I think the choice I made provides enough knowledge regarding the effects of space discretization in obtaining accurate results.

5 Conclusions

The finite difference method at second order effectively approximates the theoretical results for the first 10 eigenvalues and eigenfunctions of the Quantum Harmonic Oscillator (QHO) in natural units. These results were obtained on a discretized interval ranging from -10 to

10 with 1000 points. The method demonstrates ideal agreement with theoretical predictions in terms of the difference in energy and overlaps of the functions.
The choice of how the discretization is provided is crucial for accurately simulating higher energy levels and achieving the desired precision. A finer grid or extended domain may be required for higher energy states.
To have a correctly working simulation and facilitate future use, it is essential to adopt best practices in code development, as the ones discussed in Section 3.3. In this way the likelihood of errors is greatly reduced and the long-term usability of the same code for other tasks is provided.

References

[1] D. J. Griffiths, *Introduction to Quantum Mechanics*, 3rd Edition, Pearson, 2018.

[2] S. Montangero *Introduction to Tensor Network Methods*, Springer, 2018.