Numerical Solution for the 1D Quantum Harmonic Oscillator

Abstract

The aim of this project was to solve numerically the 1D quantum harmonic oscillator taking in account the effects of varying computational parameter in accuracy, stability and time cost of the algorithm.

1 Theory

The Hamiltonian operator of the 1D quantum harmonic oscillator has the following form.

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

where $\hat{p} = -i\hbar\frac{d}{dx}$, and $\hat{x} = x$ are respectively the momentum and position operators. The associated Schrödinger equation has the following form

$$-\frac{\hbar^2}{2m}\frac{d^2\psi(x)}{dx^2} + \frac{1}{2}m\omega^2x^2\psi(x) = E\psi(x) \tag{2}$$

To tackle this problem in a numeric fashion we can rewrite the Schrödinger equation as a linear system of equation

$$-\frac{\hbar^2}{2m}\frac{\psi(x_{i+1}) - 2\psi(x_i) - \psi(x_{i-1})}{\epsilon^2} - \frac{1}{2}\omega^2 m x_i\psi(x_i) = E\psi(x_i) \tag{3}$$

$$\big(-\frac{\hbar^2}{2m\epsilon^2}T + \frac{1}{2}m\omega^2X\big)\vec{\psi} = E\vec{\psi} \tag{4}$$

where T is is a tridiagonal matrix with elements $T_{ii} = -2$ and $T_{i,i+1} = T_{i,i-1} = 1$ while X is a diagonal matrix of elements $X_{ii} = x_i$

2 Code Development

The code first computes the Hamiltonian matrix of the discretized system. This task is performed using two functions which compute the matrices T and X and then return their value multiplied by a constant given as an input. All the other parameter requested for the computation of the

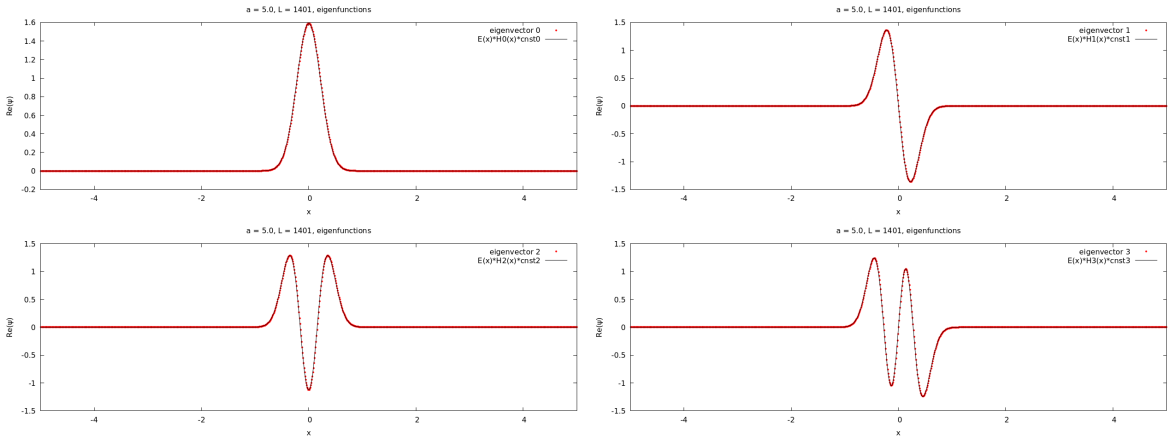

Figure 1: Graph showing the first 4 eigenvectors of the Hamiltonian and their analytical solutions as function of the position

matrices are stored as intrinsic variables in the module in which the previous functions are defined.

```
! compute kinetic energy term
function K_1d(cnst) result(K)
    implicit none
    integer :: N,ii
    double complex :: cnst
    double complex :: K(2*dim+1,2*dim+1)
    K= (.0,.0)
    do ii = 2, 2*dim
           K(ii,ii) =(-2.0,.0)
           K(ii+1,ii)=(1.0,.0)
           K(ii-1,ii)=(1.0,.0)
    end do
    K(1,1) =(-2.0,.0)
    K(2*dim+1,2*dim+1) =(-2.0,.0)
    K(2,1)=(1.0,.0)
    K(2*dim,2*dim+1)=(1.0,.0)
    K=cnst*K*(1/e**2)
end function K_1d
```

```
!intrinsic parameters:
!e side of mesh's cell
!dim = (L-1)/2
!L is the number of cells
!compute the potential energy
function x2_1d(cnst) result(x2)
    implicit none
    integer :: ii
    double complex ::
        x2(2*dim+1,2*dim+1), cnst
    double complex :: tmp
    x2 = (0.0,0.0)
    do ii = 1, 2*dim+1
        tmp%im=0.0
        tmp%re = e*(ii-1-dim)
        x2(ii,ii) = tmp**2
    end do
    x2=cnst*x2
end function x2_1d
```

The program continues summing the two matrices obtaining the full Hamiltonian and then performs the computation of its eigenvalues and eigenvectors. This computation is done using an helper function that ease the call of the lapack subroutine **zheev**, it is called **eigz**, from matmulqic

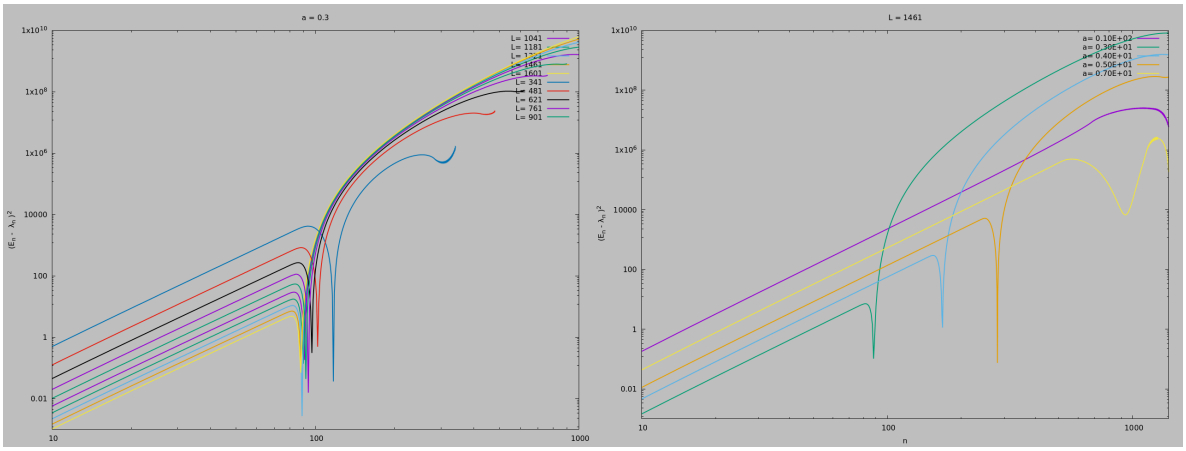

Figure 2: Graph showing $(E_n - \lambda_n)^2$ as function of n, in log-log scaling. On the left I varied L keeping the domain constant, in fact different lines represent different L. On the right I kept L fixed and increased the domain, different lines represent different domains. We can see that on the right the threshold for the exponential growth is almost the same for all the lines, while for the second is different.

module. To recover the values of the eigenfunctions of the associated Schrödinger equation it is then necessary to properly normalize the eigenvectors. This is done by calling the **norm2** function and then dividing each eigenvector by the parameter $\sqrt{\epsilon}$. This last procedure in fact recover the normalization condition of the eigenfunction $\int_{-\infty}^{+\infty} \bar{\psi}(x)\psi(x)dx \approx \sum_i \bar{\psi}(x_i)\psi(x_i)\epsilon^2 = 1$ in contrast the output of **norm2** is normalized as follows $\sum_i \bar{\psi}_{norm2}(x_i)\psi_{norm2}(x_i) = 1$.

Once a numerical representation of the eigenfunctions is obtained, the last step consisted in evaluating the performances of the algorithm in terms of reliability and time consumption. This have been done performing various tests, varying the number of cells of the mesh and their domain, measuring the execution time for each case. In particular I chosed $x \in [-a, a]$ with $a = \{3, 4, 5, 7, 10\}$ as domains, and $L = 70 \cdot i + 100$ with $i \in \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$ as number of cells.Then I computed the eigenvalues and eigenvectors for each couple of values (a, L). I also computed $\delta_n = (E_n - \lambda_n)^2$ to investigate if the precision of the results varied as function of n. Moreover I introduced two descriptive quantities $s_{20} = \frac{1}{20^2}\sum_{n=1}^{20}(E_n - \lambda_n)^2$ and $s_{300} = \frac{1}{300^2}\sum_{n=1}^{300}(E_n - \lambda_n)^2$ to better understood the overall accuracy of the results. (where $E_n = \hbar\omega(n - \frac{1}{2})$ and λ_n the computed eigenvalues).

3 Results

All the result derives from the Hamiltonian presented in the theory section using as parameters $\omega = 20, \hbar = m = 1$. A visual picture of the agreement between the theoretical eigenfunctions and the data obtained from the numerical approach can be seen in figure 1.

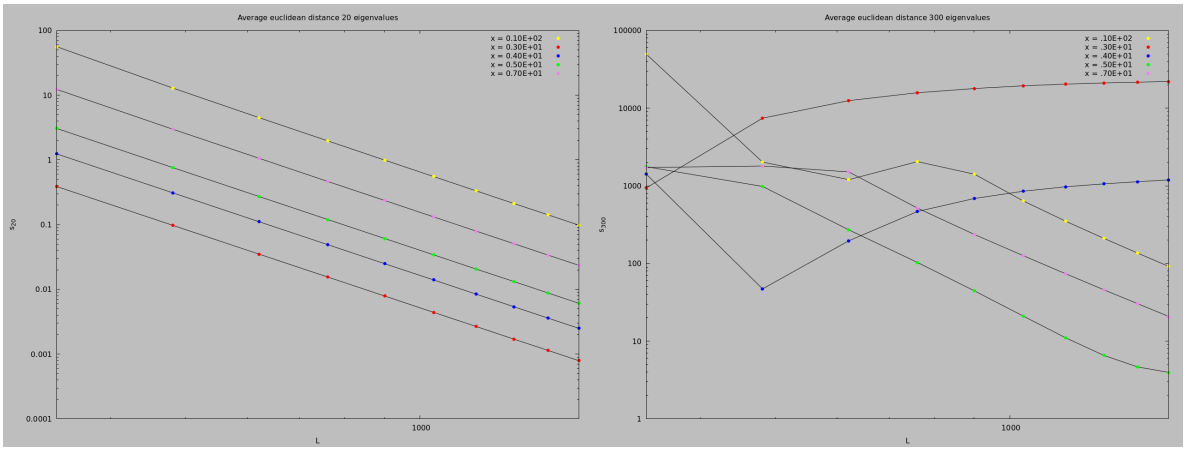

Figure 3: Graph showing on the left the s_{20} parameter, and on the right s_{300}, both as function of L, different lines represent different domains ($x \equiv a$). On the left s_{20} decreases as function of L while on the right s_{300} shows an undefined behaviour.

3.1 Stability and Precision

Evaluating the δ_n we can notice that it increases exponentially while increasing n, that means that we are more precise for low eigenvalues. The exponential growth highlights also that the relative error with respect to E_n increases, since E_n is linear in n.

The exponential growth eventually stops when n reaches a threshold value $\tilde{n}$. As we can see in figure 2, $\tilde{n}$ does not vary as function of the precision parameter, but depends on the size of the simulation domain.

Probably the deviation from the exponential growth is caused by the fact that the eigenvectors with $n > \tilde{n}$ does not approach zero inside the boundaries of the domain.

Computing s_{20} and s_{300} reinforce this thesis. As long as s_* is computed over eigenvalues with $n < \tilde{n}$ (s_{20} case) , s_* decreases as function of L, and visually seems that this decrease is exponential [Fig.3(left side)]. While when s_* is computed also for $n > \tilde{m}$ as in the s_{300} case, the behaviour as function of L is undefined [Fig.3(right side)].

From the graph showing s_{20} we can also infer about the advantages and disadvantages of increasing the precision: In fact s_{20}, as said decreases exponentially while increasing the L with fixed boundaries, which means that increasing significantly L will not affect much the precision .

3.2 Computational Time

The time necessary for the computation scale as $O(L^{3.158})$. This evaluation have been done performing a linear fit on the logarithm of the computational time as function of the logarithm of the number of cells used in the mesh [Fig.4].

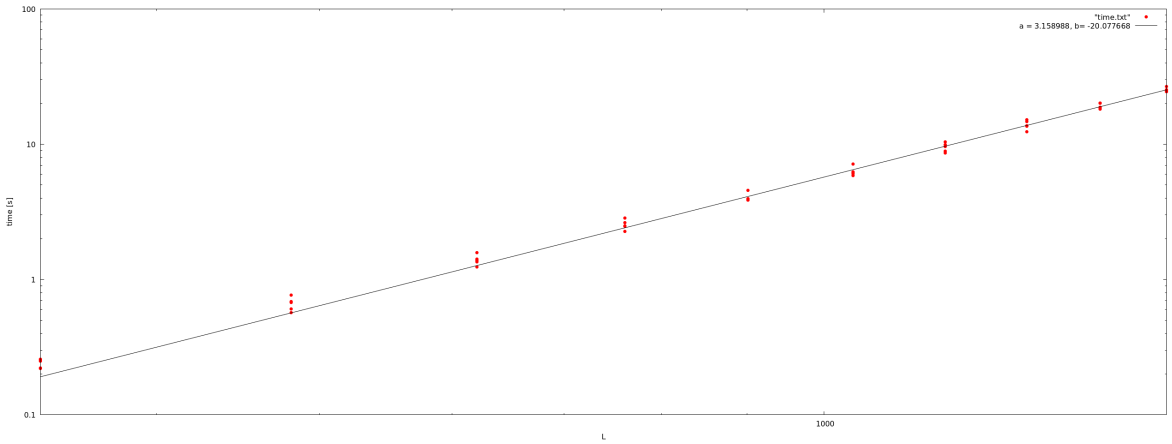

Figure 4: Graph showing the computational time as function of L the number of mesh cells

4 Self Development

The code developed performed correctly, a great improvement would be to find an analytical expression for $\tilde{n}$. This would allow to know in advance the domain requested to compute a generic eigenvalue and its respective eigenvector, and tune then the precision at will.