UNIVERSITY OF PADUA
PHYSICS AND ASTRONOMY DEPARTMENT

Homework #6
Continuous time independent Schrödinger equation

Student name: *Beatrice Segalini* – 1234430

Course: *Quantum Information and Computing 2020* – Professor: *S. Montangero*
Due date: *Novembre 17th, 2020*

Abstract

In this report, the solution of a one-dimensional time-independent Schrödinger equation is displayed. The system considered is the famous quantum harmonic oscillator, a `Fortran` program is implemented to compute eigenvalues and eigenvectors of its Hamiltonian.

Theory

Theoretical background and solution. The Hamiltonian of the one dimensional quantum harmonic oscillator is:

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

where $\hat{p} = -i\hbar\frac{\partial}{\partial x} \implies \hat{p}^2 = -\hbar^2\frac{\partial^2}{\partial x^2}$ is the momentum operator and $\hat{q} = x$ is the position one. In this work, the constants are set to simplify the problem equal to $m = \frac{1}{2}$, $\hbar = 1$, $\omega = 1$.

The time-independent Schrödinger equation for this system is:

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

where E_n denotes a to-be-determined real number that will specify a time-independent energy level, or eigenvalue, and the solution $|\psi\rangle$ denotes that level's energy eigenstate.

Equation 2 can be written in space-representation, knowing that $\langle x \mid \psi\rangle = \psi(x)$, as:

$$\left(-\frac{\partial^2}{\partial x^2} + x^2\right)\psi_n(x) = E_n\psi_n(x) \tag{3}$$

where the constants are already simplified. It is a partial differential equation dependent only on x, whose solution is well known:

$$E_n = 2n + 1 \qquad\qquad n = 0, 1, 2\ldots \tag{4}$$

$$\psi_n(x) = \frac{1}{\sqrt{2^n\,n!}} \cdot \left(\frac{1}{2\pi}\right)^{1/4} \cdot e^{-\frac{x^2}{4}} \cdot H_n\left(\sqrt{\frac{1}{2}}x\right) \tag{5}$$

The pedix n points out the quantisation of energies and it holds that $n \in \mathbb{N}$; the functions H_n are the Hermite polynomials $H_n(z) = (-1)^n\, e^{z^2}\frac{d^n}{dz^n}\left(e^{-z^2}\right)$.

Computational technique implemented. In order to solve this problem with a computational approach, one needs at first to discretise correctly the workspace.

For this purpose, a symmetric interval $[-x_{max}; x_{max}]$ of width L is selected and divided into N small intervals, each of equal width $\Delta x = \frac{2x_{max}}{N}$, obtaining a vector of $N + 1$ positions:

$$x_i = -x_{max} + (i-1)\Delta x, \text{ with } i = 1, \dots, N+1 \text{ and } x_{N+1} = x_{max} \tag{6}$$

Indicating with the pedix i the function ψ computed in the $i-$th position on this discretised grid, one can calculate the second derivative with the **finite differences method**:

$$\psi_i'' = \frac{1}{\Delta x^2}\left(\psi_{i+1} + \psi_{i-1} - 2\psi_i\right) \tag{7}$$

As a consequence, the discretised Hamiltonian of the system $\hat{H}_d$ can be written as a $(N+1) \times (N+1)$ *tridiagonal matrix* whose elements are:

$$\hat{H}_d(i,i) = \frac{2}{\Delta x^2} + \omega^2\left(-x_{max} + (i-1)\Delta x\right)^2 \tag{8}$$

$$\hat{H}_d(i,j) = -\frac{1}{\Delta x^2} \tag{9}$$

$$\hat{H}_d(j,i) = -\frac{1}{\Delta x^2} \tag{10}$$

Thus, the problem is finally reduced to finding eigenvalues and eigenvectors of this discretised Hamiltonian $\hat{H}_d$.

Code development

The code relies on two main components: a user-defined module, `quantum_util`, for performing the lattice and Hamiltonian discretisation, and the `LAPACK` subroutine `DSTEMR`.

Computation of the lattice. The discretisation of spaces is implemented with the following code (Listing 1):

Listing 1: Function to calculate the lattice.

```
    function GenLattice(xmax, xmin, N) result(grid)

integer                              :: N
double precision, intent(IN)         :: xmax, xmin
double precision                     :: dx
double precision, dimension(N+1)     :: grid

dx = (xmax - xmin)/N

do aa = 1, N+1
        grid(aa) = xmin+(aa-1)*dx
end do

end function
```

The code uses Equation 6 to generate the $N+1$ dimensional grid, ranging from `xmin` to `xmax` (then chosen as opposites), with N intermediate equal steps of width `dx`.

Computation of the Hamiltonian. To compute the Hamiltonian, a subroutine is written (Listing 2). The diagonal and subdiagonal elements of the Hamiltonian are calculated according to Equations 8, 9, 10 and stored in two different vectors: this is necessary for the application of the `LAPACK` subroutine used to derive eigenvalues and eigenvectors.

Notice that it is possible to provide a value of ω, but in this analysis this feature was not exploited.

Listing 2: Subroutine to perform the calculation the Hamiltonian non-zero elements.

```
    subroutine H_d(lattice, omega, D, E)

double precision, dimension(:), intent(IN)  :: lattice
double precision, intent(IN)                :: omega
integer                                     :: N
double precision                            :: dx
double precision, dimension(:), intent(OUT) :: D, E

N = size(lattice)

dx = lattice(2) - lattice(1)

do aa= 1, N
        D(aa) = 2d0/(dx**2) + (omega*lattice(aa))**2
        E(aa) = -1d0/(dx**2)
end do

end subroutine
```

Computation of eigenvalues E_n and eigenvectors. `DSTEMR` computes selected eigenvalues and eigenvectors of a real symmetric tridiagonal matrix, whose diagonal elements are noted as `D` and subdiagonal as `E`. The spectrum in this case is computed in a range of indices `IL=1:kk` for the desired eigenvalues.

In lines $2-7$, the LAPACK subroutine computes the optimal workspace size. After that, the arrays `evl`, `evct` are updated with the eigenvalues and eigenvectors in ascending order, if the computation converged (i.e., if the parameter `INFO` is equal to 0).

Listing 3: Code for calculating eigenvalues and eigenvectors.

```
!   compute optimal size of workspace
call DSTEMR('V', 'I', points, D, E, 0d0, 0d0, IL, kk, kk, &
                evl, evct, points, kk, ISUPPZ, TRYRAC, &
                WORK, -1, IWORK, -1, INFO)

LIWORK = int(IWORK(1))
LWORK = int(WORK(1))

!   compute eigenvalues and eigenvactors

call DSTEMR('V', 'I', points, D, E, 0d0, 0d0, IL, kk, kk, &
                evl, evct, points, kk, ISUPPZ, TRYRAC, &
                WORK, LWORK, IWORK, LIWORK, INFO)
```

Results

As one could expect, the results change significantly when the system dimensions are varied (range of positions, number of points considered). So, the eigenvalues analysis is performed on two different range of positions: $[-5, 5]$ ($L = 10$) and $[-25, 25]$ ($L = 50$); for each of them, datasets are collected with a number of intervals $N = [100, 200, 1000, 2000, 10000]$.

Eigenvalues. The first step is to check what is the highest possible number of eigenvalues that is affordable to compute by the algorithm. To do so, for each combination of N and L, the eigenvalues

E_n are plotted in a log-log plot versus their order n and compared with the theoretical value $E_{n,th} = 2n + 1$. The results are shown in Figures 1, 2.

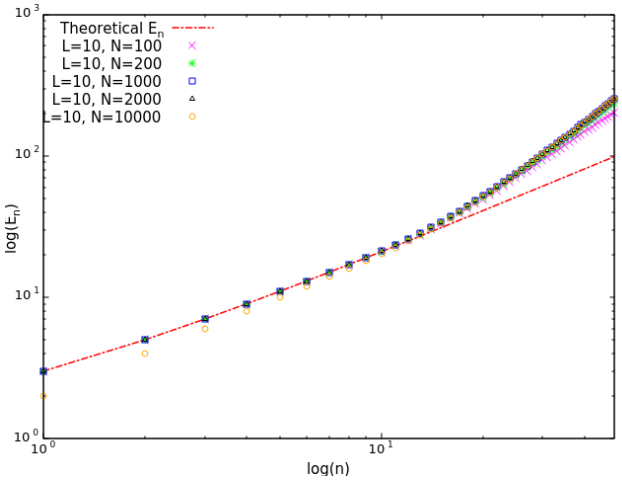

Figure 1: Eigenvalues E_n vs. their order n, space range $[-5, 5]$, with different Ns.

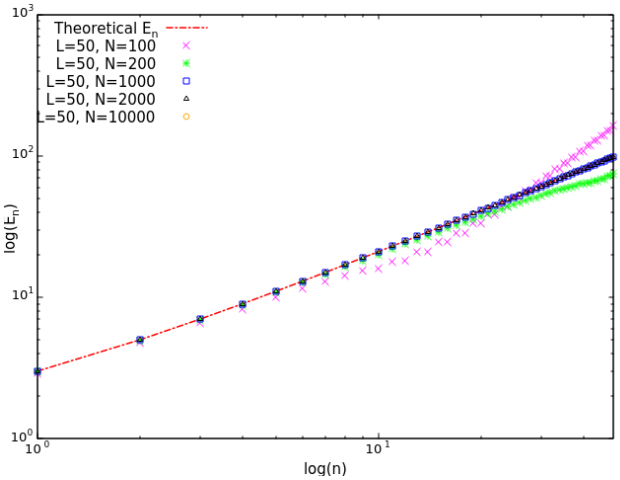

Figure 2: Eigenvalues E_n vs. their order n, space range $[-25, 25]$, with different Ns.

Both graphs show a diverging tendency with respect to the theoretical values in the rightmost part, meaning that there is in fact a computational limit in our algorithm, that depends on the system shape.

To have more insight about this and quantitatively compare the results, the relative difference:

$$Diff(E_n) = \frac{|E_n - E_{n,th}|}{E_{n,th}}$$

is plotted as a function of n, for both cases. Results are displayed in the log-log plots in Figure 3, 4.

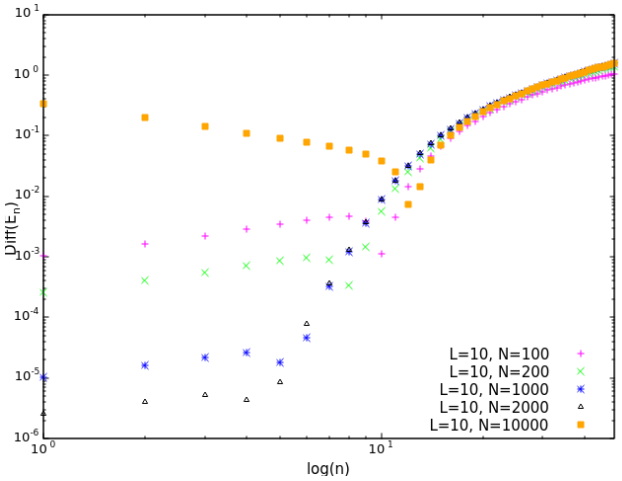

Figure 3: $Diff(E_n)$ vs. n, space range $[-5, 5]$, with different Ns.

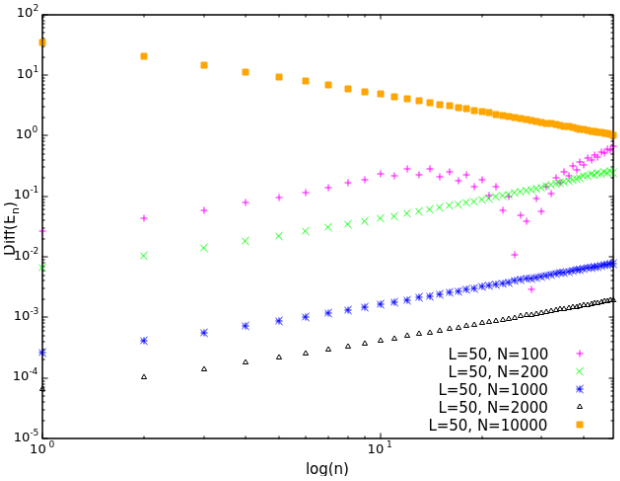

Figure 4: $Diff(E_n)$ vs. n, space range $[-25, 25]$, with different Ns.

Observing these plots, the most striking feature is definitely the increasing tendency of the relative difference, meaning that the computation becomes less and less precise when increasing the order of the eigenvalues.

In particular, for the smaller $x-$range, already after about 10 eigenvalues the relative difference overcomes the 1%.

Some of the plotted curves have, at a certain point, a change of tendency: this is due to the fact that the absolute value of the difference between the theoretical and numerical value is considered for visualising correctly in the log plot all the points.

The curve that has the worst performance is, on both graphs, the one with the bigger N considered: the error in the computation of the first eigenvalues is huge in both cases, symbol that a discretisation that is too thin lead to terribly wrong results.

The same holds when considering intervals that are too wide: in fact, the magenta and green lines of both graphs show bigger values of $Diff(E_n)$ with respect to the blue and black lines.

So, as a matter of facts, one needs to consider a *tradeoff* between the discretisation parameter N and the size of the system L.

Another plot is realised for observing a wider range of eigenvalues in the case of the best performing set of parameters, to which the value of $N = 5000$ is added (Figure 5).

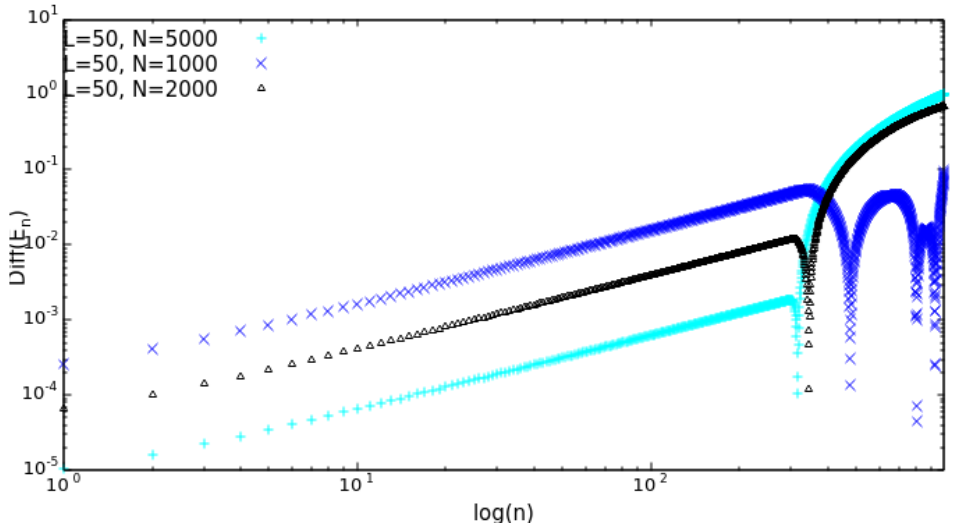

Figure 5: $Diff(E_n)$ vs. n, $L = 50$, with wider range of n considered, for $N = 5000, 1000, 2000$.

In this case, it can be noticed that the relative difference explodes or starts to behave weirdly only after more than 100 eigenvalues computed, improving significantly with respect to before. In particular, the combination $N = 5000$, $L = 50$ can compute with error $< 0.1\%$ up to 300 eigenvalues: as a consequence, when studying the eigenfunctions this set of parameters is preferred to the others.

Eigenfunctions. As already mentioned, the study of the eigenfunctions is performed with a system with $x \in [-25, 25] \implies L = 50$, and with $N = 5000$ intervals. A `Python` script is used to compute the theoretical value for the eigenfunctions, following formula 5. The eigenfunctions ψ_0, ψ_{99} computed both analytically and numerically are represented in Figures 6, 7.

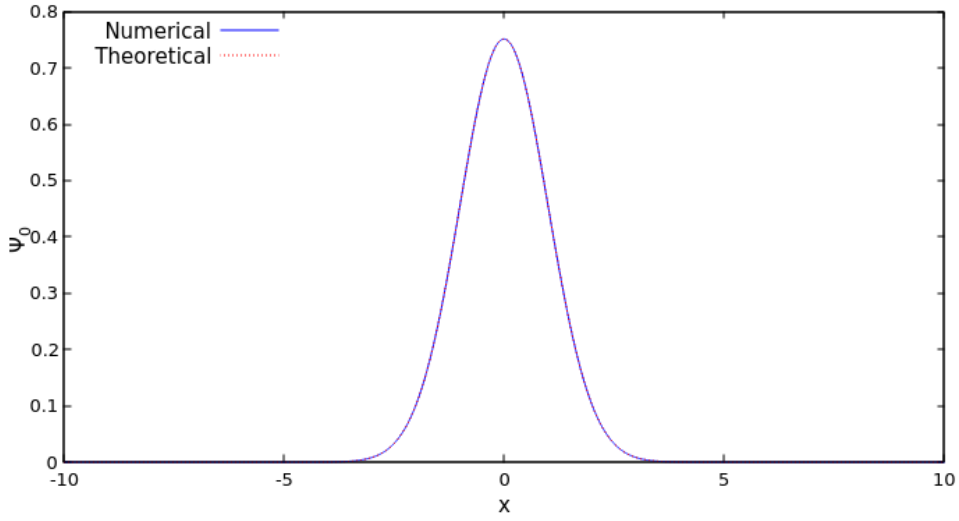

Figure 6: First eigenfunction, computed numerically and analytically. The $x-$range is adjusted for better visualisation.

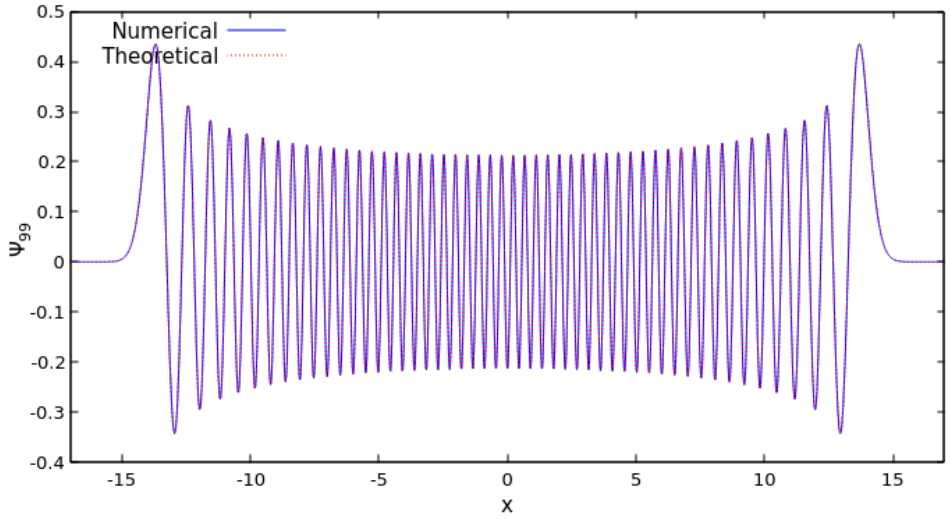

Figure 7: 99^{th} eigenfunction, computed numerically and analytically. The $x-$range is adjusted for better visualisation.

However, this kind of visualisation does not allow a clear interpretation of the goodness of the results. Therefore, similarly to before, the relative difference:

$$Diff(\psi_n) = \frac{\psi_n - \psi_{n,th}}{\max(\psi_{n,th})}$$

is used to assess the quality of the computation. This quantity is plotted versus the spacial dimension for both eigenfunctions in Figures 8, 9.

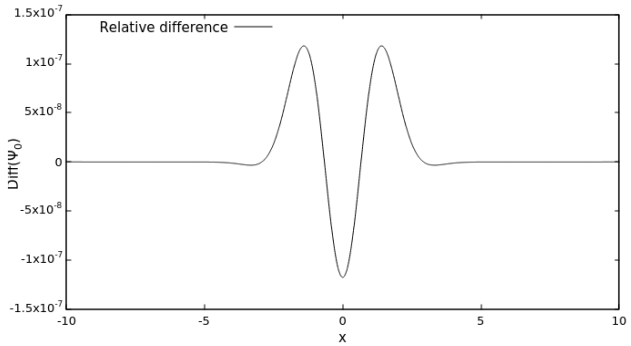

Figure 8: $Diff(\psi_0)$ vs x.

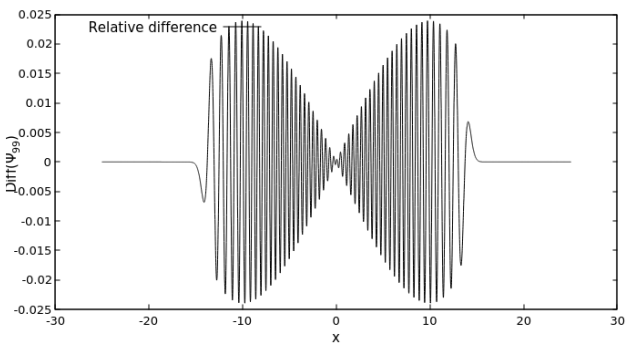

Figure 9: $Diff(\psi_{99})$ vs x.

As one can see, the first eigenfunction is computed very precisely (fluctuations are of the order of 10^{-7}). On the other hand, for ψ_{99} one can see a very symmetric tendency: the differences are enhanced in the area around ± 10, while the central part of the $x-$range has a very low error. In any case, the maximum relative difference is about 2.5%, meaning that the computation is nevertheless enough precise.

Self evaluation

Correctness. The agreement of the numerical results with the exact solution seems to only depend on the appropriate choice of the lattice: one can deduce that the algorithm implemented is able to correctly solve the time-independent Schrödinger equation.

Stability. The core of code and the most problematic parts are provided by a `LAPACK` subroutine, meaning that one can assume that the algorithm is stable and optimised.

Accurate discretisation. In this report, it was proven that good choice of the numerical domain lattice is required to get accurate results. In particular, the code written works better for bigger space domains ($L = 50$), and with a middle number of points used for the discretisation $\approx 10^3$.

Flexibility. The produced code allows an immediate customization of the lattice parameters. It can be adapted to solve other Schrödinger equations, by varying the parameter `omega`. However, the debugging procedure had not been completely implemented (despite the subroutine being there) and the main lattice parameters need to be inserted by hand, with little more work it can be significantly improved especially from this point of view.

Efficiency. The code compiles and runs pretty fast even without optimisation flags. The most time-consuming section of the code is the `LAPACK` subroutine `DSTEMR`, that nonetheless is chosen appropriately for the purpose of the exercise: it is optimised for tridiagonal matrices with real entries in double precision and computes only the requested eigenvalues/eigenvectors.