Quantum Information and Computing

prof: *Simone Montangero*

Report of exercise 6

Time-independent Schrödinger equation

Alberto Bassi

November 16, 2020

Abstract

The aim of this exercise is to solve numerically the time-independent Schrödinger equation for the 1-D harmonic oscillator by using the finite difference method.

Theoretical introduction

The Hamiltonian for the 1-D harmonic oscillator reads

$$\hat{H} = \frac{\hat{p}^2}{2m} + \frac{m\omega^2\hat{q}^2}{2} = -\frac{\hbar^2}{2m}\frac{d^2}{dx^2} + \frac{m\omega^2 x^2}{2} \,, \tag{1}$$

where in space representation the impulse and position operators are $\hat{p} = -i\hbar\frac{d}{dx}$ and $\hat{q} = x$ respectively. The time-independent Schrödinger equation reads $\hat{H}\,|\psi_n\rangle = E_n\,|\psi_n\rangle$, where the eigenvalues are $E_n = \hbar\omega\left(n + \frac{1}{2}\right)$ for $n = 0, 1, \ldots$, and the eigenvectors are, in space representation

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

where H_n are the Hermite Polynomials.
For simplicity, in the following we will work in unit of mass $m = 1$ and $\hbar = 1$[1]. Our goal is to solve this problem numerically, by using the finite difference method, in a finite symmetric interval $[-x_{max}, x_{max}]$. Therefore, in space representation the Schrödinger equation becomes $-\frac{1}{2}\psi''(x) + \frac{1}{2}x^2\psi = E\psi$.
Given $N+1$ points, the size of the discretization is $h = \frac{2x_{max}}{N}$. We define $x_i = -x_{max} + (i-1)h$ for $i = 1, 2, \ldots, N+1$ and $\psi_i = \psi(x_i)$. The finite difference second derivative reads $\psi_i'' = \frac{1}{h^2}\left(\psi_{i+1} + \psi_{i-1} - 2\psi_i\right)$. Thus, in matrix representation the discretized Schrödinger equation becomes $H\vec{\psi} = E\vec{\psi}$. Namely

$$\frac{1}{h^2}\begin{pmatrix} 1 + x_1^2\omega^2h^2/2 & -1/2 & 0 & \cdots & & 0 \\ -1/2 & 1 + x_2^2\omega^2h^2/2 & -1/2 & 0 & & \vdots \\ 0 & -1/2 & 1 + x_3^2\omega^2h^2/2 & -1/2 & 0 & \\ & & & & & 0 \\ \vdots & & & & \ddots & -1/2 \\ 0 & \cdots & & 0 & -1/2 & 1 + x_{N+1}^2\omega^2h^2/2 \end{pmatrix}\begin{pmatrix} \psi_1 \\ \psi_2 \\ \\ \vdots \\ \\ \psi_{N+1} \end{pmatrix} = E\begin{pmatrix} \psi_1 \\ \psi_2 \\ \\ \vdots \\ \\ \psi_{N+1} \end{pmatrix} , \tag{3}$$

where we have imposed that our eigenfunctions are 0 at the boundary.
Therefore, this problem reduces to compute the eigenvalues and eigenvectors of the matrix H. Precisely, we will implement the matrix h^2H in order not to deal with to high numbers while diagonalizing it. The algorithm will give us the normalized eigenvectors and the eigenvalues multiplied by h^2, so we will have to divide by the same quantity to get the correct ones. A rapid check on the correctness is to consider the parity of eigenfunctions. Since the Hamiltonian commute with the parity operator, the parity of ψ_n is $(-1)^n$.

[1] In order not to work with too small numbers.

Code Development

We have transferred the debugging module in "modules.f". The debugging variable will be initialized by the program debugging in "debugging.f" and called by the python script "exe.py". As usual, we install by "install.py". In the same module file ("modules.f"), we have written the module **quntum**, which includes the subroutine **harm_matrix** to initialize the harmonic potential matrix h^2H and the function **norm** to compute the integral of our eigenfunctions and check if they are properly normalized. These subroutines are reported below. If needed, it is straightforward to modify the potential to consider other quantum problems.

```
subroutine harm_matrix(matrix,NN,xmin,omega,hh,status)
      implicit none
      real, dimension(:,:), allocatable :: matrix
      integer :: NN,status
      real :: xmin,omega,hh,xx
      integer :: ii !iterator
      allocate(matrix(NN+1,NN+1),stat=status) !allocate matrix, n+1 points
      matrix=0.0                !off diagonal terms
      do ii=1,NN+1
         xx=(xmin+(ii-1)*hh)*hh*omega
         matrix(ii,ii)=1+0.5*(xx**2)
         if(ii.lt.NN+1) then
            matrix(ii,ii+1)=-0.5
              matrix(ii+1,ii)=-0.5
         endif
      enddo
      end
```

```
real function norm(matrix,NN,num,hh)
      implicit none
      integer :: NN,num
      real, dimension(NN+1,NN+1) :: matrix
      integer :: ii
      real :: hh
      norm=0
      do ii=1,NN+1
         norm=norm+(matrix(ii,num)**2)
      enddo
      norm=norm*hh
      return
      end
```

The program **harm_oscillator** in "main.f" reads the parameters (debug, printer, number of points, interval, pulsationω and number of eigenvectors to print in a file) from the respective temporary files in the folder temp. We use single precision.

```
program harm_oscillator
      use debugging
      use quantum
      implicit none
      !definitions
      real :: xmin,xmax              !Interval
      real :: omega !pulsation
      integer :: NN             !number of points of the disretization
      real, dimension(:,:), allocatable :: matrix !Our hermitian matrix
      real :: hh    !delta x
      logical :: debug, printer !logical variable for debugging
      integer :: status         !allocation status
      integer :: count          !counting errors
      integer :: kk !how many eigenvectors
      integer :: ii,jj !iterators
      !dsyev variables
      integer :: lda,lwork,info
      real, dimension(:),allocatable :: w,work
      real :: xx
      real :: norma
      !program start
      count=0 !error counter
      open(10,file="temp/debug.txt",status='unknown') !read parameters from temp files
      .
      .
      .
      close(13)
      hh=(xmax-xmin)/NN*1.0 !define h
```

Then it initializes the harmonic matrix h^2H,allocates the vectors needed by Lapacks's **ssyev** and computes eigenvalues and eigenvectors. Every time it prints checkpoints on the correctness of the execution (i.e. allocation, matrix dimensions, info of ssyev etc.).

```
      !Initialize harmonic potential matrix and check allocation
      call harm_matrix(matrix,NN,xmin,omega,hh,status)
      call check_allocation(status,debug,printer)
      !allocate and check ssyev variables
      allocate(w(NN+1),stat=status)
      call check_allocation(status,debug,printer)
      lda=max(1,NN+1)
      lwork=max(1,3*NN+2)
      allocate(work(lwork),stat=status)
      call check_allocation(status,debug,printer)
      !Compute eigenvalues and eigenvectors
      call ssyev('V','U',NN+1,matrix,NN+1,w,work,lwork,info)
      call check_eigen(info,debug,printer,count)
```

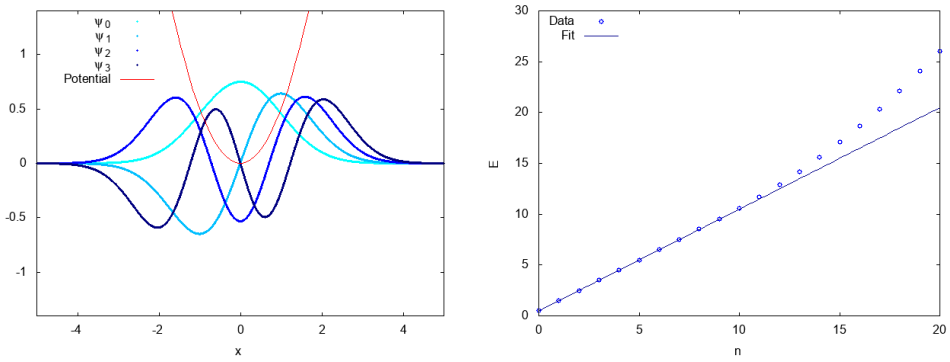
After dividing the matrix by $\sqrt{h}$(this ensures the correct normalization) the program prints eigenvalues and desired eigenvectors in a file. In the end, it checks the normalizations and frees the memory.

```
      !Divide the eigenfunctions by \sqrt{h} because ssyev returns orthonormalized
    eigenvectors
      matrix=matrix/sqrt(hh)
      !Print and save eigenvalues and eigenvectors in a file
      open(15,file="data/eigenvalues.txt",status='unknown')
      open(16,file="data/eigenvectors.txt",status='unknown')
      do  ii=1,NN+1
          xx=xmin+(ii-1)*1.0*hh
        write(16,*)xx,matrix(ii,1),matrix(ii,2),(matrix(ii,jj), jj=3,kk)
         write(15,*) ii-1,w(ii)/(hh**2)
      enddo
      close(15)
      close(16)
      !check the normalizations
      do ii=1,kk
      print *,"Norm of state",ii-1,":",norm(matrix,NN,kk,hh)
      enddo
      !Free memory and print number of errors encountered
      deallocate(matrix,w,work)
      print *, "Error encountered:", count
      end
```

Results

We have tried different combinations of x_{max}, ω and N to monitor the robustness and correctness of the algorithm. In order to check the correctness of the algorithm, we have computed the norms of the eigenfunctions, we have fitted the eigenvalues with the curve $E = \omega(n+s)^2$ and we have plotted the eigenfunctions computed numerically with the theoretical eigenfunctions in Equation 2. For each simulation[3], we have computed the first 6 eigenvectors. Our best simulation is reported in Figure 1 with parameters $(N = 2000, x_{max} = 5, \omega = 1)$, where we have included only the first 4 eigenvectors due to the legibility. From various simulations we have noticed that the larger the x interval is, the more similar the numerical solutions are to the theoretical ones. That can be seen from the residual plots of the eigenfunctions $n = 4, 5$ in Figure 2. We can see that the accordance is good, but the errors are systematic and residuals are not uniform. For the intervals we have chosen, we see that the discretization is good, we do not need more points. Moreover, the a posteriori check on the normalization, here not reported due to the limited space, succeeded.
The results of fitting the first 10 eigenvalues are reported in Table 1. They are compatible with expected values ($\omega = 1$,$s = 1/2$). Due to the finite precision, however, the eigenvalues escape the expected curve pretty soon. Moreover, by increasing the x interval such that the eigenfunctions "see" the potential, more higher order eigenfunctions become more reliable and more eigenvalues are in accordance to the expected values. In order to exploit the dependence of the correctness of the algorithm on the product $\omega\Delta x$, we have run different simulations with the same $\omega = 1$. In Figure 3 some plots are reported. We clearly notice that the bigger the x interval is, the better the numerical results accord to the theoretical curves.

[2]We remind that we imposed $\hbar = 1$.
[3]Performed by MSI Leopard Pro(2017).

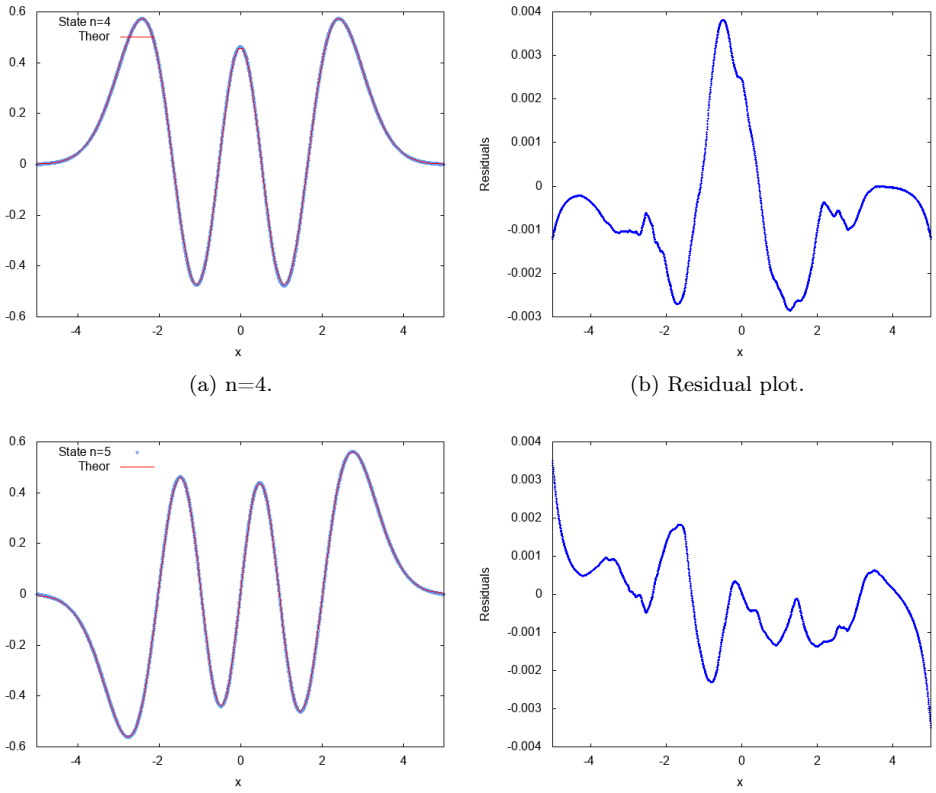


(a) First 4 eigenfunctions. (b) Eigenvalues.

Figure 1: Simulation with $\omega = 1$, $x_{max} = 5$ and $N = 2000$.

ω	1.006 ± 0.002
s	0.49 ± 0.01

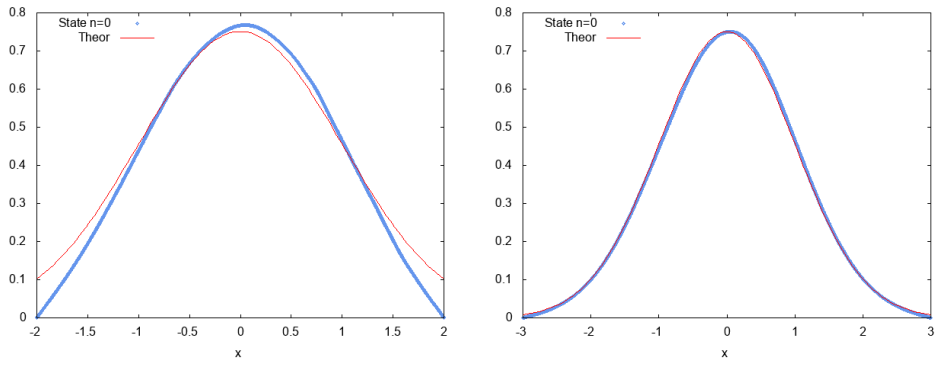
Table 1: The fit parameters for the first 10 eigenvalues.



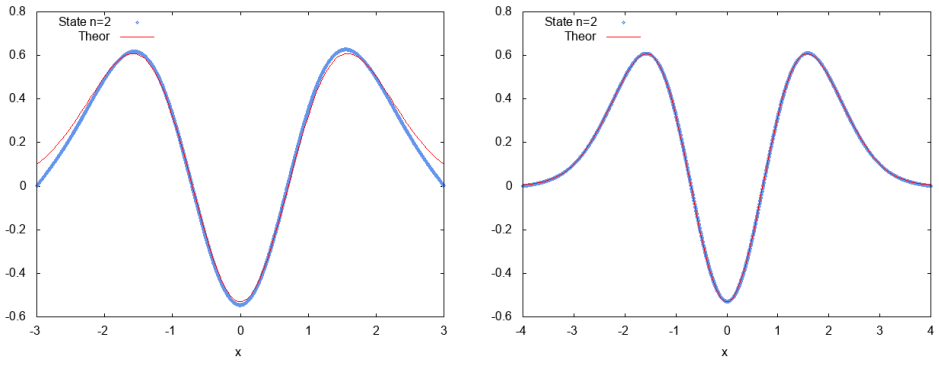
(a) n=4. (b) Residual plot.

(c) n=5. (d) Residual plot.

Figure 2: States $n = 4, 5$ for $N = 2000$, $\omega = 1$ and $x_{max} = 5$.



(a) $x_{max=2}$ (b) $x_{max=3}$



(c) $x_{max=3}$ (d) $x_{max=4}$

Figure 3: A $N = 2000$, $\omega = 1$ simulation with different intervals for the ground state and the state n=2.

Self Evaluation

In this week, we have learned how to solve numerically the time-independent Schrödinger equation. We have learned how to check the correctness of a numerical algorithm and the importance to choose the right interval and the right discretization.

From considerations of previous section, we have noticed that the accurateness of the numerical solution increases as the interval chosen increases, because of the fact that the eigenfunctions have to "see" the potential. From a few simulations with constant $x_{max} = 1$, but different $\omega < 1$, clearly emerged this fact. In fact, as $\omega \to 0$, the expected solutions approx the ones of a free-particle, while we were still imposing solutions to vanish at the boundaries, which is good only if the interval is sufficiently large. However, due to finite precision, we will always commit some error in higher order eigenfunctions and eigenvalues. We can state that the larger we take the interval, the more precise our solutions are. To improve the understanding of the problem, one could also plot the sum of square residuals in function of the product $\omega\Delta x$ (we expect it to be greater than 1 for correctness and stability) by keeping h constant. In fact, by increasing the interval we must pay attention to increase the number of points taken or, equivalently, to reduce h as well, despite computation performance. Though, our choice worked well.

With regards to the correctness, we have introduced in the program many checkpoints to control the correct allocation of the memory, the correct evaluation of the eigenvalues and so on and so forth. Frome the physical point of view, we have checked the correctness by means of the normalization, the eigenvalues and the theoretical eigenfunctions.

With regards to the stability, we have noticed that for different simulations, the algorithm gave back the same eigenfunctions, but with random sign. So, every time we needed to correct the sign in order to match our definition of eigenfunctions in Equation 2. However, these choices do not influence the physical results, since eigenfunctions are defined unless an arbitrary phase (in the real space it is a sign) and we are interested only on the square modulus of them.

With regards to flexibility, it is easy to modify the fortran subroutines to consider other potential and quantum problems. So the code is very flexible. Furthermore, one could also write a matrix initialization for a generic potential, that can be written in another module. Since the matrix is tridiagonal and symmetric, it is not necessary to save it entirely in memory, but one vector is sufficient (one diagonal is composed by only -1/2). With that in mind, the diagonalization algorithm must take it into account for further improvements.