Time dependent Schrödinger equation:

One dimensional harmonic oscillator

Abstract

This exercise asks to solve numerically the time dependent Schrödinger equation. The hamiltonian considered is again the one of the quantum harmonic oscillator, this time with a term that depends on time in the quadratic potential. The problem is one dimensional. In particular, the time evolution of the ground state $|\psi_0\rangle$ is computed, and the results for different parameters are shown.

Theory

The Hamiltonian considered in the exercise is

$$\hat{H} = \frac{\hat{p}^2}{2} + \frac{(\hat{q} - q_0(t))^2}{2} \quad with \quad q_0(t) = \frac{t}{T}, \;\; t \in [0, T] \tag{1}$$

One can notice that, respect to the Hamiltonian of the previous exercise, m and ω are set to 1. This $\hat{H}$ has a kinetic and a potential part, that depend only on the momentum operator $\hat{p}$ and the position operator $\hat{q}$, respectively.

$$\hat{H} = \hat{T} + \hat{V} \tag{2}$$

The time dependent Schrödinger equation has the following form:

$$i\frac{\partial}{\partial t}\psi(x,t) = \hat{H}(t)\psi(x,t) \tag{3}$$

and the time evolution of the eigenstate $\psi(t)$ over a time interval Δt has the form

$$|\psi(t + \Delta t)\rangle = exp(-i\hat{H}\Delta t)|\psi(t)\rangle \tag{4}$$

The constant $\hbar$ is fixed to 1. Now, we know that, in general, given two operators $\hat{A}$ and $\hat{B}$ this relation holds

$$e^{\hat{A}+\hat{B}} \sim e^{\hat{A}} e^{\hat{B}} e^{[\hat{A},\hat{B}]} \tag{5}$$

and that $\hat{V}$ and $\hat{T}$ do not commute. But for Δt *sufficiently* small one can approximate and write

$$exp(-i\hat{H}\Delta t) \sim exp\left(-i\frac{\hat{V}\Delta t}{2}\right) exp(-i\hat{T}\Delta t) exp\left(-i\frac{\hat{V}\Delta t}{2}\right) + \mathcal{O}(\Delta t^3) \tag{6}$$

In the computation we can express both $\hat{V}$ and $\hat{T}$ as two diagonal matrices. In fact the momentum operator can be represented by a diagonal matrix in the p space and the same holds for the position operator in the x space. To switch from x to p representation of a state one can apply the Fourier transform to that state

$$|\psi_x(t + \Delta t)\rangle = exp\left(-i\frac{V(x,t)\Delta t}{2}\right) \mathcal{F}^{-1}\left[exp(-iT(p)\Delta t)\mathcal{F}\left[exp\left(-i\frac{V(x,t)\Delta t}{2}\right)|\psi_x(t)\rangle\right](p)\right](x) \tag{7}$$

Computationally, as we will see in the next section, the library `FFTW` for computing the discrete Fourier transforms (DFT) allows us to express the operators in the diagonal form.

Code development

Four `MODULE`s are used to solve the task:

- `"DEBUGMOD"`, a module to debug a generic code, present also in previous exercises

- "FOURIER", that contains the file "fftw3.f03" and the iso_c_binding intrinsic module. This MODULE has been developed looking at this link [1]
- "SOLVE_AO", a module used in the previous exercise, that contains subroutines and functions to solve the one dimensional time independent Schrödinger equation through the finite difference method
- "GRID1D", a module that contains tools to create the discretized grid of points

Respect to the previous assignment the MODULE "SOLVE_AO" contains, in addiction, a SUBROUTINE "FIRST_K_EG", that performs the diagonalization through the subroutine ZHEEVX provided by LAPACK. This subroutine computes the first k eigenvalues and eigenvectors of a given matrix, in an approximated way. In this exercise k is fixed to 1, since only the evolution of the ground state is required, but one can see the evolution of the other eigenstates simply changing this value.
Moreover the main program, called EX7, asks the user to decide the size of the grid, NN, the extremes of the grid x_low and x_high, the number of eigenvalues and eigenvectors k, the number of time steps and the T value, to allow the change of the parameters of the problem quickly. At each run the program asks also the parameters ω and the mass, but in this specific exercise they are fixed to one. An important part in the program is the discretization of the grid. This is performed according to the listing 1, where "GRID_POINTS" is a SUBROUTINE, contained in the MODULE "GRID1D", shown in listing 2.

```
grid = GRID_POINTS(NN,x_low,x_high) ! create a grid of points
step = grid(3)-grid(2)
pgrid= GRID_POINTS(NN, 0.d0, (2*pi)/step*(NN-1)/NN ) ! grid for the momentum
```

Listing 1: Discretized grid

```
FUNCTION GRID_POINTS(npnt,lower,upper)  RESULT(ltc)
! this function takes as INPUTS:
! npnt = an INTEGER that represent the number of points
!                          we want in our grid
! lower & upper = extremes points, DOUBLE PRECISION
! OUTPUT: ltc = an array of dimension npnt, containing all the points
  REAL*8, DIMENSION(npnt) :: ltc
  INTEGER*4 :: npnt,ii
  REAL*8 :: lower,upper,step
  step = (upper-lower)/(npnt-1) ! the step
  DO ii=1,npnt
     ltc(ii)=lower+(ii-1)*step
  END DO
END FUNCTION GRID_POINTS
```

Listing 2: "GRID_POINTS" SUBROUTINE

In the main program the "DEBUGMOD" is used to check if the norm of the state is one before and after applying the evolution operators. We expect so, given that that evolution operator is unitary. Precisely, the check is performed in the way shown in listing 3: if the difference between the square norm of the state and 1, in absolute value, is greater of equal to 10^{-4} a warning message is printed. The bound is not *exaclty* zero since the DFT, applied to the eigenvectors to pass from x to p representation, introduce an error.

```
! check initial norm
CALL DEBUG(debug_flag,egvecs(:,1),ABS(SUM(ABS(egvecs(:,1)*SQRT(step))**2)-1) >=
  (10E-4),"state not normalized!")
```

Listing 3: The norm check

In the end the program produces four .dat files. All of them contain in the first column the 'spacial' grid that goes from x_low to x_high, the selected extremes. The number of columns corresponds to the selected NN, the points of the grid. In the other columns the "realpart.dat" file contains the real part of the wave function in each selected time step and the "imaginarypart.dat" file contains the imaginary part. With a similar structure the file "potential.dat" contains, in the other columns, how the potential evolves at each time steps, while "prob.dat" contains how the probability density associated to the wave function (*the modulus square*), evolves. These files are used by the GNUPLOT scripts for producing animated .gif, shown in the next section.

[1] http://fftw.org/doc/Overview-of-Fortran-interface.html

Results

In this section the animated `.gif`s that show the time evolution for the potential, the modulus square, the real and the imaginary part of the wave function are discussed. The chosen parameters are the following: size of the grid $NN = 70$, ω=mass=k=1, number of time steps = 80, extremes of the grid x_low and $x\_high \pm 5$. Keeping fixed these parameters, the value of T is changed, as the exercise requires. The results are shown for $T = 20$, $T = 35$ and $T = 45$. Looking at the `.gif`s (`"animateT20.gif"`,`"animateT35.gif"` and `"animateT45.gif"` in this folder), that show 50 frames of the process from $t = 0$ to $t = 50$ (in steps of 2), one can state that the wave function oscillates when the quadratic potential moves. Moreover when T increases the velocity of the `.gif` decreases, as expected from a dimensional analysis ($\frac{1}{T}$ has to be a velocity, since it is multiplied by a time in the subtraction to the position $\hat{q}$ in the hamiltonian 1). In the figures below I report for each T the frame corresponding to that T for which $t = T$. We can observe that the potential, that for $t = 0$ has the minimum in $x = 0$, for $t = T$ has the minimum in $x = 1$.

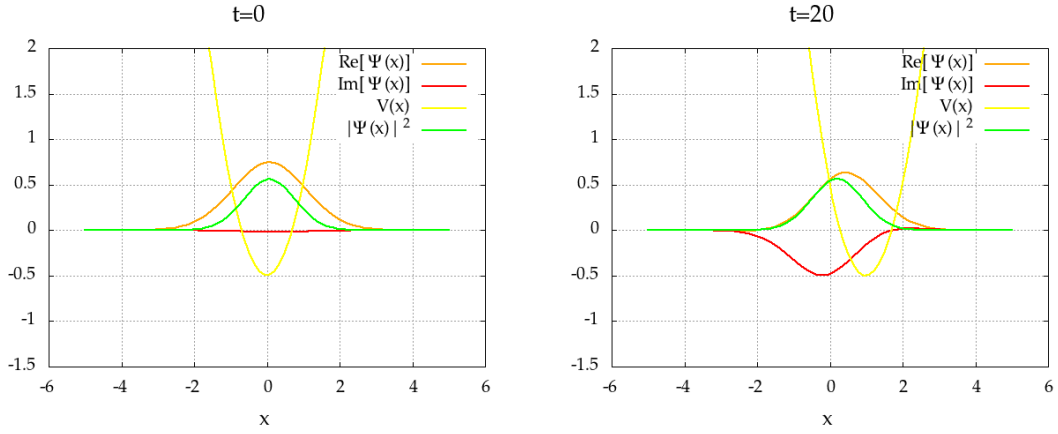

Figure 1: Case for $T = 20$. Two frames are shown, for $t = 0$ and for $T = t$.

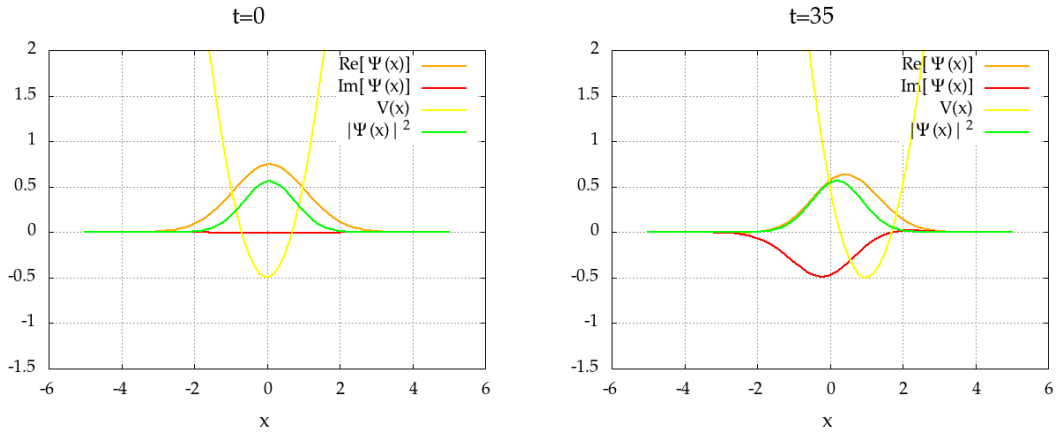

Figure 2: Case for $T = 35$. Two frames are shown, for $t = 0$ and for $T = t$.

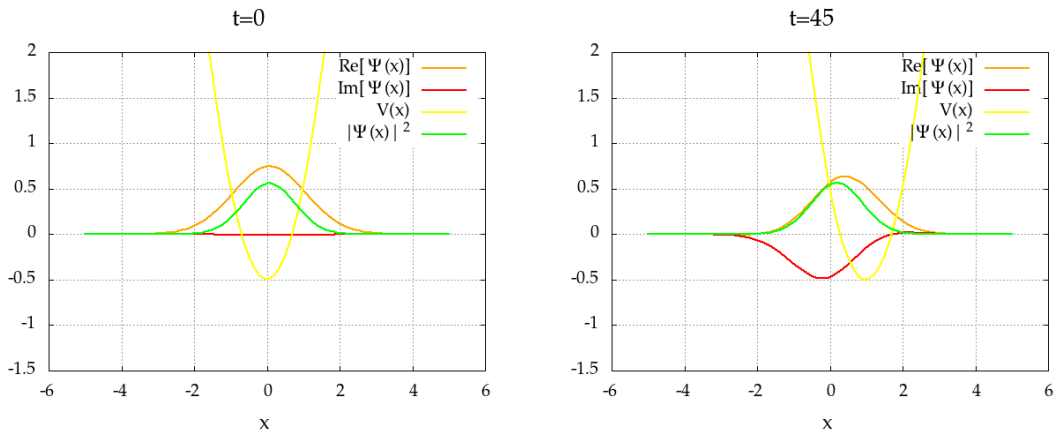

Figure 3: Case for $T = 45$. Two frames are shown, for $t = 0$ and for $T = t$.

Self-Evaluation

In this exercise I discovered the library `FFTW`, that in this case provides a huge computational advantage. I spot also that the so called *split operator method* is a powerful approximation that allows to write equation 6. Moreover I learnt how to create animated `.gif` in `GNUPLOT` for a fancy representation of the time evolution.