Quantum Information and Computing

prof: *Simone Montangero*

Report of exercise 7

Time-dependent Schrödinger equation

Alberto Bassi

November 23, 2020

Abstract

The aim of this exercise is to solve numerically the time-dependent Schrödinger equation with the split-operation method for a 1-D harmonic potential travelling at constant velocity.

Theory

Let us consider the time-dependent Schrödinger equation

$$i\hbar\frac{\partial|\psi\rangle}{\partial t} = \hat{H}|\psi\rangle \ , \qquad \hat{H} = \frac{\hat{p}^2}{2} + \frac{(\hat{x} - x_0(t))^2}{2} \ , \tag{1}$$

where $x_0(t) = t/T$ is the minimum of a harmonic oscillator potential travelling at constant velocity $v = 1/T$ and $|\psi(0)\rangle = |0\rangle$ is the ground state of harmonic oscillator.

Algorithm

Since the hamiltonian operator $\hat{H}$ can be written as a sum of two terms

$$\hat{H} = \hat{T}(\hat{p}) + \hat{V}(\hat{x}, t) \ , \tag{2}$$

which are respectively the kinetic energy operator and the potential operator, we can use the split-operation method to solve numerically Equation 1. From Baker-Campbell-Hausdorf (BCH) formula, the time evolution operator can be decomposed as ($\hbar = 1$)

$$e^{-i\hat{H}\Delta t} = e^{-i(\hat{T}+\hat{V})\Delta t} \overset{BCH}{=} e^{-\frac{i\hat{V}\Delta t}{2}} e^{-i\hat{T}\Delta t} e^{-\frac{i\hat{V}\Delta t}{2}} \ , \tag{3}$$

where we are neglecting terms of Δt^3. Then, by taking into account that the kinetic operator and the potential operator are diagonal in momentum and space representation respectively, we can write the time evolution of the wave-function as

$$|\psi(x, t + \Delta t)\rangle = e^{-\frac{i\hat{V}\Delta t}{2}} \mathcal{F}^{-1}\left[e^{-i\hat{T}\Delta t}\mathcal{F}\left(e^{-\frac{i\hat{V}\Delta t}{2}}|\psi(x,t)\rangle\right)\right] \ , \tag{4}$$

where $\mathcal{F}$ is the Fourier Transform (FT) operator, which allows to go from space to momentum representation. What we can implement is the discrete Fourier Transform (DFT), by first choosing a proper space and momentum discretization. Let us assume to work in the space interval $[-x_{max}, x_{max}]$ and let us discretize it with $N = 2^n$[1] intervals of length $h = \frac{2x_{max}}{N}$. Therefore, at time t the DFT is

$$\psi(p_k, t) = \sum_{j=0}^{N-1} e^{-ik\frac{2\pi}{N}j}\psi(x_j, t) \ , \tag{5}$$

which takes values in the interval $[-\frac{\pi}{x_{max}}, \frac{\pi}{x_{max}}]$ of the momentum space. The algorithm repeat iteratively the procedure of Equation 4 in the time interval $[0, T]$ with steps of $\Delta t = T/N_t$. The implementation of Equation 5 normally scales as $O(N^2)$. But, we can improve that asymptotic behaviour by performing the Fast Fourier Transform (FFT), which requires $O(N\log(N))$ operations instead. Since for each step of the split-operation method we are summing operations which scale as $O(N)$ (since we are applying diagonal matrices), overall the computation time scales as $[3 * O(N) + 2 * O(N\log(N))] * O(N_t) = O(N_t N\log(N))$.

[1] This choice is done to improve the performances of FFT.

Theoretical expectation

The evolution of an operator under an hamiltonian $\hat{H}$ in Heisenberg visual is given by Heisenberg equation

$$\frac{d}{dt}\hat{O}(t) = i[\hat{H}, \hat{O}] \quad (\hbar = 1) \, . \tag{6}$$

For our hamiltonian in Equation 1, it is straightforward to get the system

$$\frac{d}{dt}\hat{x} = \hat{p} \qquad \frac{d}{dt}\hat{p} = -\hat{x} + vt \, , \quad (m = \omega = 1) \, . \tag{7}$$

If we solve it and take the expectation values, by imposing the initial conditions $\langle \hat{x}(0) \rangle = \langle \hat{p}(0) \rangle = 0$ [2], we get

$$\langle \hat{x}(t) \rangle = -v \sin t + vt \, , \qquad \langle \hat{p}(t) \rangle = -v \cos(t) + v \, . \tag{8}$$

We could have foreseen it since classically this is a spring travelling at constant velocity v.
Moreover, since the initial state is a coherent state [3] we expect it to maintain the same Gaussian shape, but centred in the average oscillating value $\langle \hat{x}(t) \rangle = -v \sin t + vt$.

Code Development

The code is arranged in three .f95[4] files, stored in "code". The well-known "debugging.f95", which initialize the debugging variables, "modules.f95", which includes the modules **debugging** and **quantum** and "main.f95", which contains the main program **harm_oscillator_timedep**. Module **quantum** includes all the main subroutines we need for this problem. In particular, subroutine **harm_matrix** to initialize harmonic oscillator matrix to compute the ground state by means of Lapack's **dsyev** (widely discussed in the previous report), the subroutine **potential**, which applies the time-dependent potential diagonal matrix to a generic double complex wave-function *wave* and the other parameters needed.

```
subroutine potential(wave,NN,omega,xmin,hh,dt,time,speed)
       implicit none
       integer ::NN
       double complex, dimension(NN) :: wave
       double precision :: omega, xmin, hh,V,dt,time,speed
       integer :: ii
       do ii=1,NN
          V=0.5*((xmin+hh/2+hh*dble(ii-1)-time*speed)*omega)**2
          wave(ii)=exp(-cmplx(0d0,1d0)*0.5*V*dt)*wave(ii)
       enddo
     end subroutine potential
```

The subroutine **kinetic** to apply the diagonal kinetic matrix to a generic double complex wave-function.

```
 subroutine kinetic(wave,NN,pmax,dt)
       implicit none
       integer ::NN
       double complex, dimension(NN) :: wave
       double precision :: pmax,K,dt
       integer :: ii
       do ii=1,NN/2
          K=0.5*((pmax*dble(ii-1))**2)
          wave(ii)=exp(-cmplx(0d0,1d0)*K*dt)*wave(ii)
       enddo
       do ii=NN/2+1,NN
          K=0.5*((pmax*dble(ii-NN-1))**2)
          wave(ii)=exp(-cmplx(0d0,1d0)*K*dt)*wave(ii)
       enddo
     end subroutine kinetic
```

And the subroutine **FFT**, which computes the FFT to a generic double complex wave-function according to a given flag (1 for the forward FFT, -1 for the backward FFT). We point out that since FFTW's subroutines used do not properly normalized outputs, we need to divide them by $\sqrt{N}$, which is the dimension of our wave-functions.

```
  subroutine FFT(in,out,NN,flag,count)
       use,intrinsic :: iso_c_binding
       implicit none
```

[2] since the initial wave-function is $|0\rangle$.
[3] i.e. an eigenstate of the lowering operator a_-.
[4] we chose this extension to work correctly with library FFTW.

```
        include 'fftw3.f03'
        integer :: NN, ii, flag,count
        double complex, dimension(NN) :: in,out
        integer*8 :: plan
        if(flag==1) then
           call dfftw_plan_dft_1d(plan,NN,in,out,FFTW_FORWARD,FFTW_ESTIMATE)
           call dfftw_execute_dft(plan,in,out)
           call dfftw_destroy_plan(plan)
           out=out/sqrt(dble(NN))
        elseif(flag==-1) then
           call dfftw_plan_dft_1d(plan,NN,in,out,FFTW_BACKWARD,FFTW_ESTIMATE)
           call dfftw_execute_dft(plan,in,out)
          call dfftw_destroy_plan(plan)
          out=out/sqrt(dble(NN))
        else
           print *, "Error: Invalid Flag"
           count=count+1
        endif
      end subroutine FFT
```

To control the correctness of the results, we search for the position of the maximum of $|\psi|$ and the maximum values there attained thorough built-in subroutines.

```
subroutine peak_search(vec,NN,xmin,hh,position,maxim)
        implicit none
        integer :: NN,ii
        double complex, dimension (NN) :: vec
        real, dimension(NN) :: temp
        double precision :: xmin,hh,position,maxim
        do ii=1,NN
           temp(ii)=sqrt(dble(vec(ii)*conjg(vec(ii))))
        enddo
        maxim=maxval(temp)
        ii=findloc(temp,maxim,dim=(NN))
        position=xmin+(ii-1)*hh+hh/2
      end subroutine peak_search
```

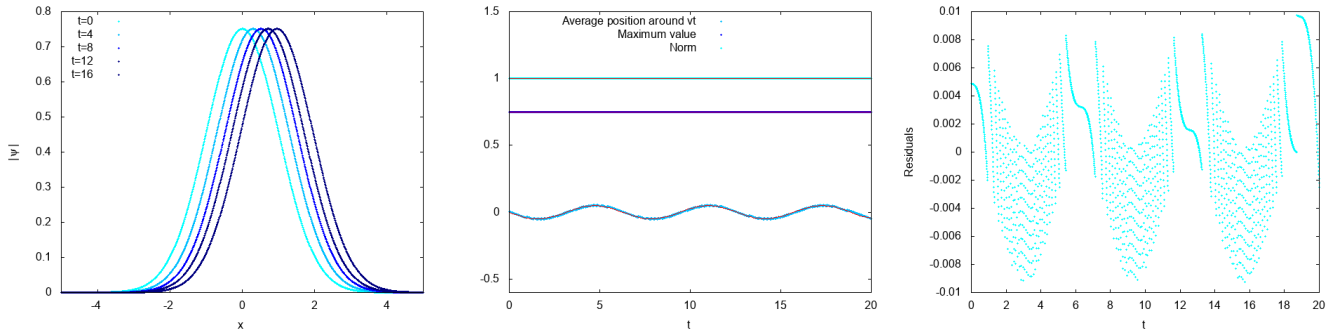
In the end, the main program **harm_oscillator_timedep** computes the time evolution of the ground state of the harmonic oscillator and saves it in a matrix to be printed later.

```
  do ii=1,N_time
          time=dt*dble(ii)
          call potential(wave,NN,omega,xmin,hh,dt,time,speed)  !compute space evolution
          call FFT(wave,temp,NN,1,count)!Forward FFT to go to p-space
          wave=temp
          call kinetic(wave,NN,pmax,dt)   !compute kinetic evolution
          call FFT(wave,temp,NN,-1,count)!Backward FFT to get back to x-space
          wave=temp
          call potential(wave,NN,omega,xmin,hh,dt,time,speed)    !Last sapce evolution
          do jj=1,NN
             ev_temp(jj,ii+2)=sqrt(dble(wave(jj)*conjg(wave(jj))))
          enddo
          call  peak_search(wave,NN,xmin,hh,position,maxim)
          write(20,*) time, norm(wave,NN,hh),position-speed*time,maxim
      enddo
```

Results

We perform a simulation[5] with parameters $(n = 11, x_{max} = 10, T = 20(v = 0.05), N_t = 2000)$, whose results are reported in Figure 1. We pay attention to choose a properly large space interval in order not to have problems with instability, a correctly thin time discretization and a sufficiently large time T in order to appreciate fluctuations of the average position around vt.
From Equation 1 we can appreciate that the wave-functions are correctly normalized, the their maximum value remains almost constant and the expectation value $\langle \hat{x} \rangle$ satisfies the theoretical prediction. The curves are all fitted with the theoretical expectation, in particular we notice that the residual plot for the sine curve is good. Fit parameters are reported in Table 1.
However, in order to appreciate better the time evolution, the reader is strongly recommended to run an interactive simulation through "animation.py", which shows the time evolution of the square modulus of the wave-function and the potential.

[5] with MSI Leopard Pro(2017).



(a) Square modulus of wave-functions. (b) Statistics. (c) Average position residuals.

Figure 1: Time evolution for a simulation with parameters $(n = 11, x_{max} = 10, v = 0.05, N_t = 2000)$.

Norm	Maximum of $\|\psi\|$	v
1±E-13	0.75 ± E-07	0.0497±0.0001

Table 1: Fit parameters.

Self Evaluation

This week we have learned how to solve numerically the time-dependent Schrödinger equation by means of the split-operation method. To do so, we have learned how to perform the FFT,which is a fundamental programming tool. However, we had to pay a lot of attention on choosing the right momentum discretization in order to get the correct results. Consequently, this week we have spent a lot of time in debugging with the hope to avoid the same errors in future.