Quantum Information and Computing 2020/21
Week 7 report

Rocco Ardino

(Dated: Monday 23rd November, 2020)

In this work we provide a numerical solution to the time dependent Schrödinger equation for a monodimensional harmonic oscillator, focusing on the time evolution of the ground state. The split operator method is employed for the purpose and the results found are visualized through a colormap plot, showing the time evolution of the probability wave.

1 Theory

The time dependent Hamiltonian of the system under study reads:

$$H(t) = \underbrace{T}_{\text{kin. term}} + \underbrace{V(t)}_{\text{pot. term}} = \frac{p^2}{2m} + \frac{1}{2}m\omega^2[x - x_0(t)]^2 \quad , \tag{1}$$

where $x_0(t) = \frac{t}{T}$, with T a characteristic parameter of the potential. Starting from the work done for the time independent version of last week assignment, we want to solve the time dependent Schrödinger equation for this type of Hamiltonian in a space interval $[-a, a]$ and in a time interval $[t_0, t_f]$. A good strategy to face the problem is to discretize both space and time intervals into N_x and N_t smaller intervals of width Δx and Δt respectively, defining the steps:

$$\begin{aligned} x_i &= -a + i\Delta x \qquad i = 0, \dots, N_x \\ t_j &= t_0 + j\Delta t \qquad j = 0, \dots, N_t \end{aligned} \quad . \tag{2}$$

Then, we solve the Schrödinger equation for the starting time t_0 through the technique exploited in the previous week assignment. Focusing on the evolution of ground state ψ_0 found through diagonalization, we can evolve this state in time through the time evolution operator:

$$\psi_0(x, t_0 + \Delta t) = e^{-i\Delta t H}\psi(x, t_0) \quad . \tag{3}$$

Using te Baker-Campbell-Hausdorff formula and approximating the commutator $[T, V(t)]$ to zero, we obtain:

$$e^{-i\Delta t H} \approx e^{-i\frac{\Delta t}{2}V} e^{-i\Delta t T} e^{-i\frac{\Delta t}{2}V} + \mathcal{O}\big((\Delta t)^3\big) \quad . \tag{4}$$

It is convenient to apply the operators in the basis where that operator is diagonal. Therefore, the Fourier transform $\mathcal{F}$ can be employed to switch from the position to the momentum space and viceversa, using $\mathcal{F}^{-1}$, leading to the main rule of split operator method for the time evolution:

$$\begin{aligned} \psi_0(x_{i+1}, t_{j+1}) &= e^{-i\Delta t H(x_i, t_j)}\psi(x_i, t_j) \\ &\approx e^{-i\frac{\Delta t}{2}V(x_i, t_j)} A_{\mathcal{F}^{-1}} \mathcal{F}^{-1} e^{-i\Delta t T(p_i)} A_{\mathcal{F}} \mathcal{F} e^{-i\frac{\Delta t}{2}V(x_i, t_j)} \psi_0(x_i, t_j) \quad . \end{aligned} \tag{5}$$

with $A_{\mathcal{F}} = A_{\mathcal{F}^{-1}} = \frac{1}{\sqrt{N_x+1}}$ normalization factors of the discrete (anti-)Fourier transform of the wavefunction. This algorithm can be applied iteratively for every time step in order to get the wavefunction solution at time t_f.

2 Code Development

For this work, the new module **tdep_se_utils** is implemented, containing several functions and subroutines for multiple purposes:

- **init_ham(N,L,M,O,t0,T)**, for instantiating the $(\texttt{N}+1)\times(\texttt{N}+1)$ tridiagonal matrix H with:
  - ▷ initial time `t0`;
  - ▷ potential characteristic time parameter `T`;
  - ▷ a symmteric space interval of length `L` for the simulation;
  - ▷ `M` and `O` values for the parameters m and ω respectively;
- **diag_ham(ham,eigs)**, for diagonalizing the tridiagonal symmetric matrix `ham` and returning the eigenvalues and eigenfunctions inside the appropriate input arguments, which are modified during the execution;
- **ham_T(N,L,M)**, to get the kinetic term of Hamiltonian;
- **ham_V(N,L,M,O,time,TT)**, to get the potential term of Hamiltonian at $t = \texttt{time}$;
- **fft(psi)**, a wrapper to the Fastest Fourier Transform in the West (FFTW3), which returns the FFT of the wavefunction `psi`;
- **afft(psi)**, as `fft` function but it does the anti-FFT to return in the position space;
- **time_ev(psi,DT,L,M,O,time,TT)**, for evolving `psi` from $t_i = \texttt{time}$ to $t_{i+1} = \texttt{time} + \texttt{DT}$;
- **print_wvfc(xs,ys,filename,unit)**, for printing on a certain file real and imaginary part of the discretized wavefunction ψ at a certain time step;
- **print_wvfc_cmap(xs,ys,zs,filename,unit)**, for printing on a certain file the discretized squared modulus $|\psi(x_i,t_i)|^2$ for every point of the space grid and for every time step, in order to get a file for colormap plot;
- **print_wvfc_expv(xs,ys,zs,filename,unit)**, for printing on a certain file the expectation value $\langle x\rangle$ for every time step.

Moreover, another module, namely **cmdline**, is implemented to have a command-line argument parser and for default arguments handling. In particular, we explain the new core functionalities of both the modules in the following subsections.

2.1 (Anti-)Fast Fourier Transform wrapper

The code to perform the discrete FFT of the wave function is showed in Listing 1. This is nothing more than a wrapper of the **fftw3** functions for FFT, for which a plan is usually defined, then executed and finally destroyed.

```
function fft(psi) result(FFT_psi)
    ! input arguments
    complex(8), dimension(:) :: psi

    complex(8), dimension(size(psi,1)) :: FFT_psi
    integer(4) :: N
    integer(8) :: plan

    N = size(psi,1)

    call dfftw_plan_dft_1d(plan, N, psi, FFT_psi, -1, 64)
    call dfftw_execute_dft(plan, psi, FFT_psi)
    call dfftw_destroy_plan(plan)
end function fft
```

Listing 1. Implementation of the wrapper of `fftw3` FFT functions.

On the other side, the code to perform the discrete anti-FFT (AFFT), and so to return the wavefunction in position space, is showed in Listing 2. The implementation is analogous to the one written for FFT wrapper.

```
function afft(psi) result(AFFT_psi)
    ! input arguments
    complex(8), dimension(:) :: psi

    complex(8), dimension(size(psi,1)) :: AFFT_psi
    integer(4) :: N
    integer(8) :: plan

    N = size(psi,1)

    call dfftw_plan_dft_1d(plan, N, psi, AFFT_psi, +1, 64)
    call dfftw_execute_dft(plan, psi, AFFT_psi)
    call dfftw_destroy_plan(plan)
end function afft
```

Listing 2. Implementation of the wrapper of `fftw3` AFFT functions.

2.2 Time evolution through Split Operator Method

The wrappers previously described are called inside the subroutine **`time_ev`**. The latter evolves the input vector, namely the wave function, of a time step Δt, keeping track of the actual time of simulation and of the other parameters of the Hamiltonian in order to reconstruct the kinetic and potential terms. The implementation of the subroutine is showed in Listing 3.

```
subroutine time_ev(psi, DT, L, M, O, time, TT)
    ! input arguments
    complex(8), dimension(:) :: psi
    real(8) :: DT, time, TT
    real(8) :: L, M, O

    real(8), dimension(size(psi,1)) :: V
    real(8), dimension(size(psi,1)) :: T
    integer(4) :: N
    integer(4) :: ii

    N = size(psi,1) - 1

    V = ham_V(N, L, M, O, time, TT)     ! potential term
    T = ham_T(N, L, M)                  ! kinetic   term

    ! apply exponential containing the potential term
    do ii=1,N+1
        psi(ii) = ZEXP(COMPLEX(0.0d0,-0.5d0*DT*V(ii))) * psi(ii)
    end do
    ! fft of wave function and normalization
    psi = fft(psi) / SQRT(1.0d0*N+1)
    ! apply exponential containing the kinetic term
    do ii=1,N+1
        psi(ii) = ZEXP(COMPLEX(0.0d0,-1.0d0*DT*T(ii))) * psi(ii)
    end do
    ! afft of wave function and normalization
    psi = afft(psi) / SQRT(1.0d0*N+1)
    ! apply exponential containing the potential term
    do ii=1,N+1
        psi(ii) = ZEXP(COMPLEX(0.0d0,-0.5d0*DT*V(ii))) * psi(ii)
    end do
end subroutine time_ev
```

Listing 3. Implementation of the subroutine for time evolution of ψ at a time t and of a step Δt.

2.3 Main program

Lastly, all the functions described before are employed in a main program, in which a command-line argument parser is implemented in order to obtain an easier handling of the parameters of the

simulation. Moreover, this functionality is also capable of giving default values to the arguments when they are not given. To make the program more user-friendly a `-h` help option has been added and in the beginning of the execution all the arguments given are printed along with a flag: `F` if a value is given for it, `T` if default is taken. An example of this functionality and of the way to start the program is showed in Listing 4. A complete list of the options is given in Table 1.

Arg option	Arg name	Arg meaning	Default
`-NX`	`stepsX`	N_x	1000
`-L`	`lengthX`	L	10
`-NT`	`stepsT`	N_t	1000
`-T`	`lengthT`	T	5
`-TF`	`finalT`	t_f	5
`-M`	`mass`	m	1
`-O`	`omega`	ω	1
`-h`	`help`	print help	

Table 1. Command-line arguments of the program.

```
$ ./07.o -NX 2000 -NT 2000 -T 1 -TF 8

Running with the following cmdline options:

NX =         2000     default [T/F]: F
L  =      10.         default [T/F]: T
NT =         2000     default [T/F]: F
T  =      1.0         default [T/F]: F
TF =      8.0         default [T/F]: F
M  =      1.0         default [T/F]: T
O  =      1.0         default [T/F]: T
```

Listing 4. Example of launch and user interface.

After the parser, if no exceptions are found, a subroutine with the simulation is called, storing $\mathrm{Re}[\psi(x,t_j)]$ and $\mathrm{Im}[\psi(x,t_j)]$ for the j^{th} time step in the dedicated folder `results` in the format "`i`_step_wvfc.dat". This is done for every $j = 0, \dots, N_t$. Moreover, a bigger file "cmap_wvfc.dat" containing all the $|\psi(x_i,t_j)|^2$ for every i, j is saved to produce a colormap plot to visualize both space and time evolution.

Last but not least, the program must be compiled with the flag `-ffree-line-length-0` and linking the lapack and fftw3 libraries.

3 Results

The simulation is run for $N_x = N_t = 2000$, $L = 20$. The characteristic time T of the potential is set to the default value of 1, while t_0 and t_f are set to 0 and 8 respectively. Several values of the physical parameters (m, ω) are tested. The results for the evolution of $|\psi(x,t)|^2$ are showed for all the sets of (m, ω) in Figures 1, 2, 3 and 4. In all the cases, the main point to observe is that the expectation value for the position, namely $\langle x \rangle$, evolves with constant velocity towards the right side of the box, with a small sinusoidal component of oscillation. This is the behaviour expected from the given Hamiltonian with such potential.

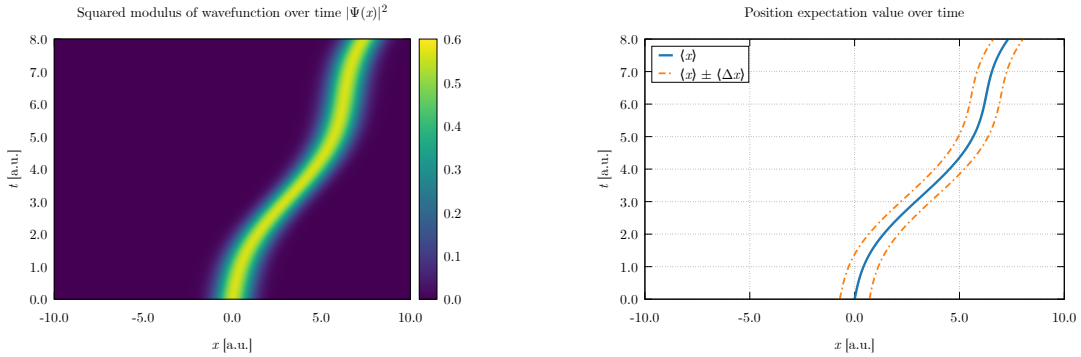

Figure 1. $(m, \omega) = (1, 1)$: colormap and $\langle x \rangle$ evolution of the probability wave $|\psi(x,t)|^2$.

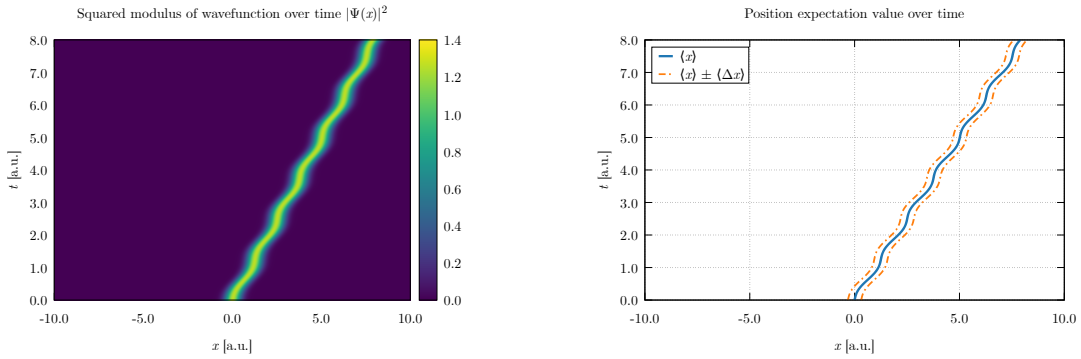

Figure 2. $(m, \omega) = (1, 5)$: colormap and $\langle x \rangle$ evolution of the probability wave $|\psi(x,t)|^2$.

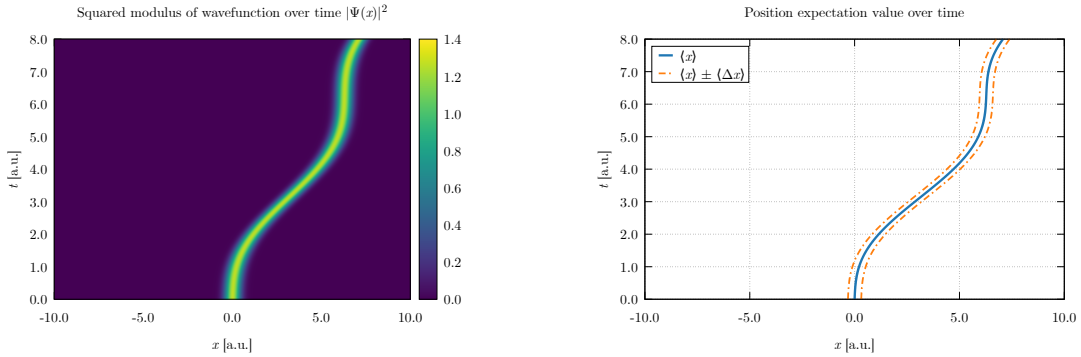

Figure 3. $(m, \omega) = (5, 1)$: colormap and $\langle x \rangle$ evolution of the probability wave $|\psi(x,t)|^2$.

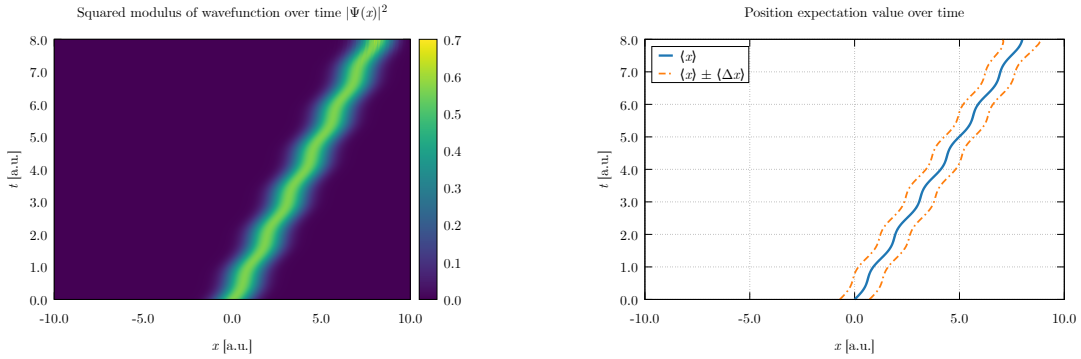

Figure 4. $(m, \omega) = (0.2, 5)$: colormap and $\langle x \rangle$ evolution of the probability wave $|\psi(x,t)|^2$.

4 Self-evaluation

The numerical simulation of the time dependent Schrödinger equation has led to successful results. We managed to work with both Lapack and FFTW3 libraries and to observe the evolution of the ground state of the given Hamiltonian, confirming the correctness of the code written. The implementation was done in a user-friendly and accurate way, introducing also an argument parser with a help functionality in order to understand the needed parameters for the simulation and their meanings.