Quantum Information and Computing
Ex. 07

Andrea Nicolai - 1233407

November 23, 2020

Abstract

In this work we solved the time dependent Schrödinger's equation for a single particle inside a harmonic potential whose minimum moves at a constant velocity. The so called "split"-operator was introduced to approximate the Hamiltonian as if it were constant for small time intervals, and finally the package *FFTW* was used to switch between momenta and position spaces. This has been done for different choices of numerical parameters, such as the space grid size dimension $[x_{min}; x_{max}]$, the spacing in time dt and space dx.

Theory

The Hamiltonian of the Schrödinger's equation for the 1-dim quantum harmonic oscillator, whose potential does depend on time, in coordinate basis is the following:

$$\hat{H} = -\frac{\hbar^2}{2m}\partial_x^2 + \frac{1}{2}m\omega^2(x - q(t))^2 \tag{1}$$

where m and ω are respectively the mass of the particle and the angular frequency of the oscillator, and where $q(t) = \frac{t}{T}, \quad t \in [0, T]$. We consider the time evolution of the ground state, namely the time evolution of:

$$|\psi_0\rangle = \left(\frac{m\omega}{\pi\hbar}\right)^{\frac{1}{4}} e^{-\frac{m\omega x^2}{2\hbar}} \tag{2}$$

We must solve the time dependent Schrödinger's equation:

$$i\hbar\partial_t|\psi_t\rangle = \hat{H}(t)|\psi_t\rangle \tag{3}$$

where the Hamiltonian is time dependent. Therefore we are not allowed to write $|\psi(t)\rangle = e^{\frac{-iE_0 t}{\hbar}}|\psi_0\rangle$, nevertheless if we discretize Δt in order to make it sufficiently small we can treat the Hamiltonian as if it were time independent and constant in such small interval. Hence we are neglecting that the kinetic $\hat{T}$ and the potential $\hat{V}$ terms generally do not commute. Formally:

$$e^{-i\hat{H}\Delta} = e^{-i(\hat{T}+\hat{V})\Delta t} \approx exp\left(\frac{-i\hat{V}\Delta t}{2}\right) e^{-i\hat{T}\Delta t} exp\left(\frac{-i\hat{V}\Delta t}{2}\right) \tag{4}$$

Note as we can exploit the fact that in momenta basis $e^{-i\hat{T}\Delta t}$ is a diagonal matrix, in order to decrease the complexity of our computations: it will scale as $O(N)$. However, if we want to switch coordinates we are requested to first compute the Fourier transform of our vector and then its inverse, which both scale in the computational "fast version" as $O(Nlog(N))$, that is still less than the normal matrix-matrix multiplication $O(N^2)$. Our "chain" of computations will be the following:

$$|\psi_0(t+\Delta t)\rangle = e^{-i\hat{V}\Delta t}\mathcal{F}^{-1}e^{-i\hat{T}\Delta t}\mathcal{F}e^{-i\hat{V}\Delta t}|\psi_0(t)\rangle \tag{5}$$

Code Development

The program can be called via a python script and accepts exactly 6 arguments, namely x_min, x_max, the spacing dx, t_min, t_max, the spacing dt. If either one is missing, it raises an error. For instance, the total interval $[x\_min, x\_max]$ is split in equally spaced intervals of dimension dx and the same occurs with time. In order to obtain the temporal evolution of our ground state, given the potential dependent on time, we recycled the code used in previous assignment to compute $|\psi_0\rangle$. In addition, to make the code more elegant we defined two new types, namely the MY_1D_GRID and $WAVE\_FUNCTION$. The first one must be initialized passing as parameters the spacing dx, x_min and x_max and it returns an array with the edges of each interval. It could be used hereafter to deal with 1-dim grids such as the space grid, time grid or the momenta grid. On the other hand, the second type stores a MY_1D_GRID element, that is the coordinate basis we want to use, and a double complex array which stores the value of the wave function at each point.
Since we will end up dealing with momenta grid, one should note that once we fixed the position 1-dim grid with its size, we are implicitly fixing the maximum momentum we obtain, namely $p_{max} = 2\frac{\pi}{dx}$. Each p_j is computed with the following logic, recalling that N is the total number of points and $j \in \{-N/2, .., N/2\}$

$$p_j = \frac{2\pi j}{N\hbar dx} \tag{6}$$

and, when initializing the grid, we shall keep into account that the library **FFTW** stores the positive momenta in the first half of the array starting from p_0, while the negative ones in the second half of the momenta grid; hence momenta coordinates must be initialized accordingly. Indeed, the Fourier transform is computed by the mean of some subroutines from **FFTW** package. The compilation flags to be added will therefore be: $-llapack - lm - lfftw3$ and we should add the $api\ fftw3.f03$ file to the directory where we are compiling.
Starting from $|\psi_0\rangle$, we may want to check whether it is normalized and start computing the time evolution for the wavefunction, moreover we need to compute the value of the potential at every time instant:

```
DO ii = 2, time_grid%size_grid
(..)
```

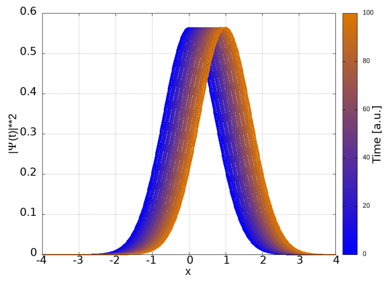
```
!START procedure
potential_vect_time = COMPUTE_REAL_POTENTIAL_TIME_T(mass,&
& omega, space_grid, time_grid%elem_ii(ii), time_interval)
potential_dt_matrix(:,ii) = potential_vect_time
temp_psi_pos_sp%values = EXP(DCMPLX(0d0, &
& -0.5d0*tt_step*potential_vect_time(:) )) * psi_0%values(:)
!Call the forward transform and then multiply
CALL dfftw_execute_dft(plan_forwardT, temp_psi_pos_sp%values, &
& temp_psi_mom_sp%values)
temp_psi_mom_sp%values = EXP(DCMPLX(0d0, &
& -tt_step*(momenta_grid%elem_ii(:)**2)*0.5d0)) * temp_psi_mom_sp%values(:)
!Normalize wrt the grid size!!
temp_psi_mom_sp%values = &
& temp_psi_mom_sp%values/SIZE(temp_psi_mom_sp%coord_1d%elem_ii)
!Call the inverse transform and then multiply
CALL dfftw_execute_dft(plan_backwardT, &
& temp_psi_mom_sp%values, temp_psi_pos_sp%values)
psi_dt%values = EXP(DCMPLX(0d0, &
& -0.5d0*tt_step*potential_vect_time(:) )) * temp_psi_pos_sp%values(:)
(..)
psi_sq_dt_matrix(:,ii) = &
& DREAL(psi_dt%values(:))**2 + DIMAG(psi_dt%values(:))**2
psi_0%values = psi_dt%values
END DO
```

In the last code we skipped the part where we initialize/destroy the plans for the forward and backward transforms for a matter of space. Note that a check was made in order to control the normalization of the norm of the wavefunction. Regarding this last sentence, we had to divide every entry by the number points in the grid for a matter of normalization, this ”problem” must be treated carefully and is introduced by the $FFTW$ package. Moreover it was added the possibility to print the wavefunction real and complex coefficient, and for each timestep, in case of it would be of our interest, despite we will not show it in result section.
Last passage we do is then to export to file in output columnwise the matrix $psi\_sq\_dt\_matrix(:, ii)$ whose name is defined according the parameters we used in the program. Every column of it stores the norm square value of the wavefunction for each point of the space grid at the ii-th time moment. A similar argument is pointed out when storing the potential values in function of time instants into another file. Out of this two data files we run some gnuplot scripts in order to create an animated gif with the time evolution and, finally, the average position and its indeterminacy wrt time.

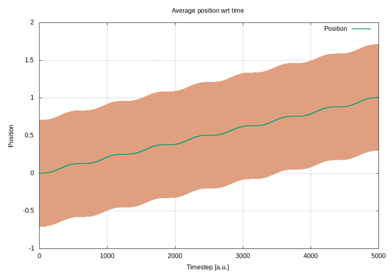
Results

We executed the program with different choices of space interval length and space resolutions. Here we will present the some of the most significant ones, namely some ”good” results, and some results that denotes numerical instability.

However, the constants m $\hbar$, ω are fixed to $1.0d0$, despite one can easily change them within the programming lines before compiling.
As a first example of correct evaluation, we chose $x_{min} = -4.0$, $x_{max} = 4.0$, $dx = 1e-2$ and time $t_{max} = 5e1$ with $\Delta t = 0.01$. The evolution of the norm of wavefunction is the following, noting that the color changes according to the time elapsed, see fig 1a. We have chosen the space grid length is enough to contain the particle wavefunction and its time evolution.



(a) Temporal evolution of the norm of wavefunction wrt time. The choice of spacings is $dt = 0.01$ and $dx = 0.01$.

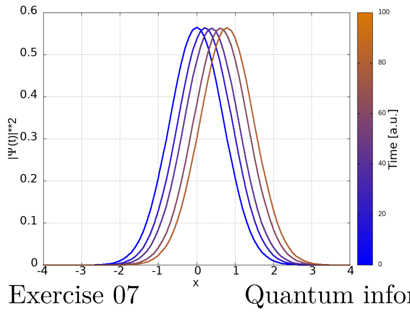


(b) Average position of the norm of the wavefunction and its time evolution. The shaded area is the indeterminacy.

In addition fig. 1b depicts how the average position of the square of wavefunction changes in time. The average position was computed by the mean of a Python script, that returned printing to a file how average position and the indeterminacy at each timestep. As one can easily see there are some points where it spends some more time. This is actually the rest point of an harmonic potential that moves at a constant velocity $1/T$, and whose position is given by t/T. The particle keeps oscillating around this point, that is moving, and therefore in turn moves itself.

Now we will discuss other results that happens with the same choice of interval length (we recall $[x_{min}; x_{max}] = [-4.0; 4.0]$ but different resolution in time and space $dx = 0.1$, $dt = 0.1$ and see whether something happens. As one can easily see from fig. 2, results are still good and no instability is noticeable, despite the larger Δt and dx. The larger choice of Δt, however, leads to a picture with a less number of square of wavefunctions shown (indeed the figure is less "filled"), but it is just a matter of layout and of how was written the gnuplot script: indeed it samples every 100 timesteps and if Δt is larger more time is span.

Figure 2: Time evolution of the norm of wavefunction for lower time and space spacings $dx = 0.1$, $dt = 0.1$. Results are good, also in this case, despite the lower number of points due to the fixed length of intervals.



The most interesting results occur when we choose a smaller length of the space grid: i.e. $[x_{min}; x_{max}] = [-2.50; 2.50]$. Indeed we start noticing some numerical instability that is both present no matter the choice of the spacing. The wave function starts to appear not gaussian-shaped anymore, but more noisy. Since we observe this behavior for both cases, we may be tempted to conclude that is due to the interval length that is not enough to contain the whole wavefunction. Indeed we see that the borders are not able to contain the whole width of the curve. One more interesting behavior is that the norm is not conserved anymore: with the help of the grid we can see how the height varies as time increases. This can be seen as well in the animations that are linked to this report: despite the potential smoothly translates, the wavefunction shows some instability.

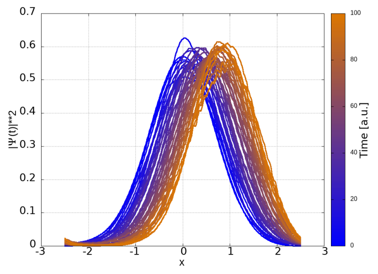


Figure 3: Temporal evolution of the norm of wavefunction wrt time. The choice of spacings is $dt = 0.01$ and $dx = 0.01$, while interval length is different: $[x_{min}; x_{max}] = [-2.50; 2.50]$.

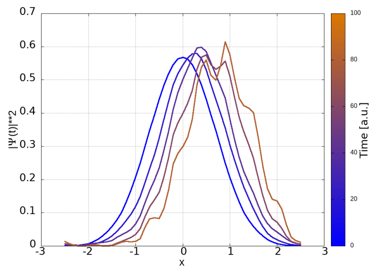


Figure 4: Temporal evolution of the norm of wavefunction wrt time. The choice of spacings is $dt = 0.1$ and $dx = 0.1$, while space interval length is different: $[x_{min}; x_{max}] = [-2.5; 2.5]$.

Self Evaluation

In this homework we have seen how to solve numerically time dependent Schrödinger's equation. This was done exploiting some mathematical properties that introduce a small error, but still approximate quite well what we would expect. The choice for the length of space grid is really important, as we can see in the result section: it must be enough large to contain the whole wavefunction and its evolution in time. We could have actually tried with a different velocity, namely chanhing t_{max} value, and see whether numerical instability would occur. Some more checks might have been implemented, such as the conservation of the square norm during each timestep: if it changes by a arbitrarily large value program would stop and raise an error due to numerical instability. Another check would have been to present some animation with the evolution of the real and complex parts of the wavefunction wrt time.