Time-dependent Schrödinger equation

Abstract

In this assignment, a time-dependent Schrödinger equation is solved numerically. The potential is quadratic, but shifting linearly with time. The equation is solved for a window of parameters, and the behaviors of the resulting solutions is then discussed.

Theory

The hamiltonian that was used is the following:

$$\hat{H} = \frac{\hat{p}^2}{2} + \frac{(\hat{q} - q_0(i))^2}{2}$$

where $q_0(i) = (T/n)t$, with $T > 0$, n as the number of time intervals to consider and i as an index, $0 \le t \le n$. This setting is slightly different from the given one; however, it is more general, and it allows to replicate the given setting (by simply taking $T = 1$). Details aside, this situation is that of a quadratic potential moving toward the positive direction of the x-axis. In order to study the time evolution of this equation, the *split-operator method* can be employed. Consider a hamiltonian

$$\hat{H} = \hat{T} + \hat{V},$$

where the kinetic and the potential parts only depend on the momentum and on the position respectively. Based on the time-dependent Schrödinger equation,

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

we can describe the time evolution of an eigenstate over a small time interval Δt as

$$|\psi(t+\Delta t)\rangle = \exp(-i\hat{H}\Delta t)\,|\psi(t)\rangle = \exp(-i(\hat{T}+\hat{V})\Delta t)\,|\psi(t)\rangle\,,$$

with $\hbar = 1$. The exponential can not be directly split: indeed, the Baker-Campbell-Hausdorff formula dictates that, given two operators $\hat{A}$ and $\hat{B}$, $e^{\hat{A}+\hat{B}} \sim e^{\hat{A}}e^{\hat{B}}e^{[\hat{A},\hat{B}]}$. In general, $\hat{T}$ and $\hat{V}$ do not commute; however, for sufficiently small values of Δt, one can write

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

effectively splitting the action of the involved operators. If $\hat{V}$ is only dependent on the position, it can be represented by a diagonal matrix in the x space; conversely, if $\hat{T}$ is only dependent on the moment, it can be represented by a diagonal matrix in the p space. Recalling that a state can be turned from x to p representation by taking its Fourier transform (and, conversely, the opposite can be done by employing the inverse Fourier transform), it is possible to significantly simplify the computations by expressing each operator in its diagonal form, applying it to the appropriate representation of the state. Explicitly, denoting the Fourier transform by $\mathcal{F}$ and its inverse as $\mathcal{F}^{-1}$,

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

The main computational advantage of this approach is that it does not rely on matrices, as diagonal operators can be effectively written as vectors. Furthermore, Fourier transforms can be efficiently computed. An important remark is that, in the case at hand, the potential is also dependent on t.

Code development

The Fortran source files are organized as follows:

- `FFTW.f90`: a module that includes the file `fftw3.f03` and the `iso_c_binding` intrinsic module. This snippet was used to avoid referencing the files directly in the main program.
- `lattice1d.f90`: this module, which was initially implemented in order to create a lattice data type, has basically become a container for a few utility functions that deal with the creation of grids of points.
- `debug_module.f90`: the usual debugging module.
- `schrod.f90`: a module which was extensively used in the previous assignment, it contains utilities for solving the 1D, time independent Schrödinger equation.
- `time_evo.f90`: the main program, where the actual computations take place and the results are produced and saved.

The solution of the time-dependent equation is structured as follows: the first step consists in computing the initial condition, that is, the eigenstates at time $t = 0$. The finite difference method that was developed in the past assignment is employed once again. However, since the ground state was the only required eigenstate for which to compute the evolution, the `LAPACK` subroutine `ZHEEVX` is used. This subroutine allows to compute an approximation for the first k eigenvalues and eigenvectors of a given matrix; this offers a great computational boost, while the loss in precision can be controlled by the tolerance parameter of the subroutine. Furthermore, employing this strategy allows the user to also study the behavior of higher-order eigenstates. After the initial eigenstates have been computed, the momentum space discretization is created. This is done by the following lines:

```
grid = LINSPACE(NN,x_low,x_high)
step = grid(2)-grid(1)
pgrid= LINSPACE(NN, 0.d0, (2*pi)/step*(NN-1)/NN )
pgrid(NN/2:)=pgrid(NN/2:)-pgrid(NN)-pgrid(2)
```

The `LINSPACE` function creates a grid of `NN` equally spaced values, ranging from `x_low` to `x_high`, extremes included. The instruction in the last line, which translates the second half of the `pgrid` vector down, is written in order to follow `FFTW`'s prescription for frequencies. Different choices lead to different results, which significantly diverge from the expected ones; more on this point later.

Two wrapper subroutines were then implemented, in order to hide the inner workings of `FFTW`'s DFT subroutines, and make the code more readable.

The debugging utilities were employed in order to perform a quick post condition check: after each evolution step has been performed, the square norm of the state is computed and compared with its expected value, which is 1. If the difference between the two is higher than some bound (which was kept at 10^{-4}), a warning is printed on screen. The choice of the bound was dictated by the fact that the computation of the DFT introduces a small error, and this propagates after each step. A stricter bound would have resulted in warnings being issued at each time step, regardless of any actual violation of the normalization due to bugs.

Finally, a gnuplot script which produces `.gif` animated images and a python script for setting the simulation parameters were implemented.

Results

Note that some animated `.gif` files are provided, which effectively highlight the evolution of the state. The first thing to notice is that the state is normalized[1] throughout the evolution. This is consistent with the theoretical expectation, as the evolution operator is unitary, and so does not modify the norm.

The following plots show the time evolution as a series of plots on the same graph, where the color represents the time index of the wave. The wave gets more and more red as time passes. A series of examples is reported; all of them are computed on a grid ranging from -10 to 10, while the other parameters (final point T, number of intermediate steps n, potential constant ω) are varied.

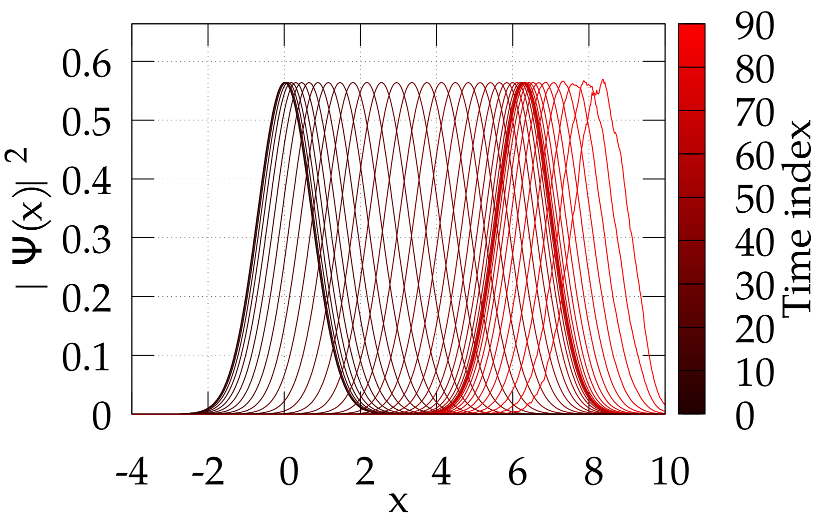

Figure 1: Parameters: $[x_l, x_h] = [-10, 10]$, $\omega = 1$, $T = 10$, $n = 100$, $N = 1024$. Plotted every 2 time steps, up to 90. Border effects become visible at this point.

[1]As previously specified, normalization is checked up to a finite margin of 10^{-4}.

All the plots show the same main characteristics (the differences appear at the borders, due to the finite grid effect). The wave is dragged to the right by the potential, without any major change in the shape of the wave itself. However, the motion of the wave itself, unlike that of the underlying potential, is not linear. As the density of the lines in the plot clearly show, the wave moves in bursts, intertwined with periods of stasis (this is clearer if one looks at the `.gifs`). Indeed, the `.gifs` show that there is a delay between the movement of the potential and the movement of the wave.

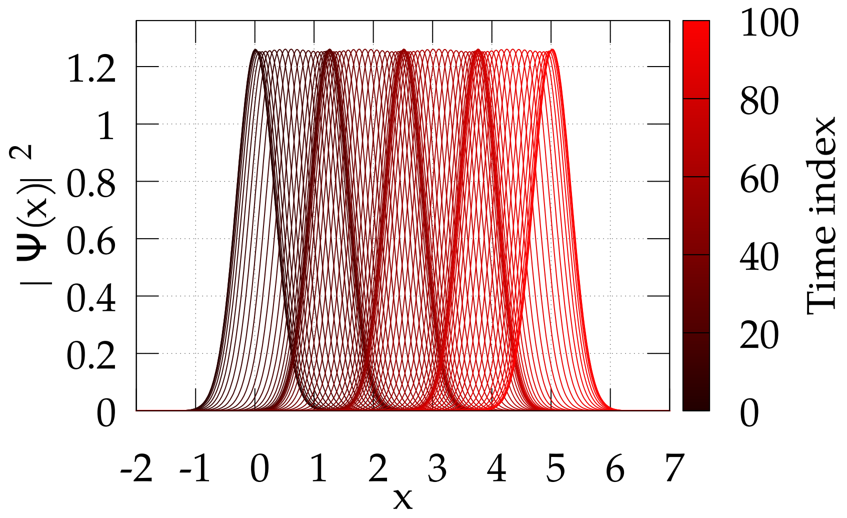

Figure 2: Parameters: $[x_l, x_h] = [-10, 10]$, $\omega = 5$, $T = 5$, $n = 100$, $N = 1024$. Plotted every time step.

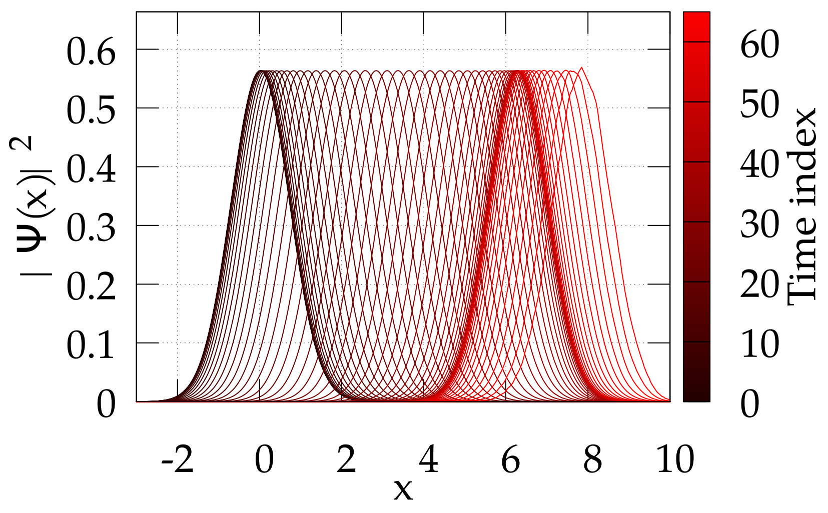

Figure 3: Parameters: $[x_l, x_h] = [-10, 10]$, $\omega = 1$, $T = 40$, $n = 300$, $N = 512$. Plotted every time step, up to 65. Border effects become visible at this point.

This delay may be a quantum effect: the ground state is not at 0 energy. The initial shift of the potential induces an increase of the potential itself on the left part of the wave, but to the central and the right part, this variation is much smaller (due to the potential being quadratic), and so almost negligible with respect to the ground energy. Indeed, when the potential is shifted according to the energy level of the ground state, one can see that the wave starts moving when the potential reaches the maximum of the wave itself.

As time passes, however, this delay results in the wave being at a quite high potential while having barely moved: the effect is a sort of ‘catch up’ process, during which the wave moves even faster than the potential. This ‘overcompensation’, in turn, results in the right part of the wave having a high potential. The process is effectively symmetrical.

Indeed, if one keeps track of the position of the maximum of the wave, and plots it against the various time instants, the following behavior is found, which is plotted against the following ansatz (which models the position of the maximum of the ground state as a function of the time index t):

$$p_{MAX}(t) = \frac{T}{n}t + \frac{1}{\omega}\sin\left(\frac{T\omega}{n}t - \pi\right)$$

The linear term comes from the potential, while the sinusoidal part models the oscillation of the wave inside the potential. The agreement is good, up until border effects appear; when the movement is slow enough, however, the discretization becomes visible.

A fitting procedure on the ansatz lead to negligible improvements (the parameters T/n and ω changed only at the third digit). In the last example (fig. 8), where the discretization becomes visible, the fitting produces a function which simply mimics the overall trend, but disregards the fact that only the points at the end of a horizontal section are meaningful (as the grid is not refined enough to better keep track of the shift of the maximum of the state).

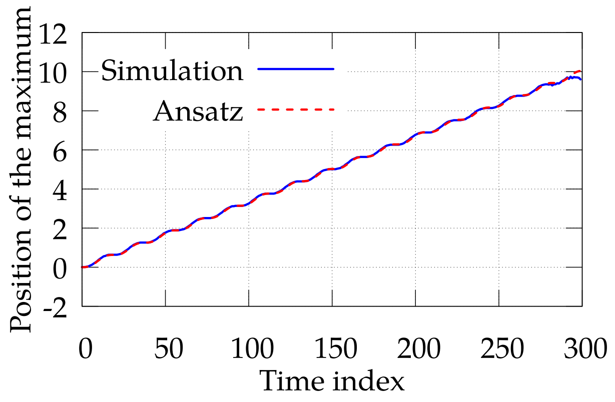

Figure 4: Location of the maximum of the ground state vs. time index. Parameters:
$[x_l, x_h] = [-10, 10]$,
$\omega = 10$, $T = 10$,
$n = 300$, $N = 1024$. Border effects become visible at the extreme.

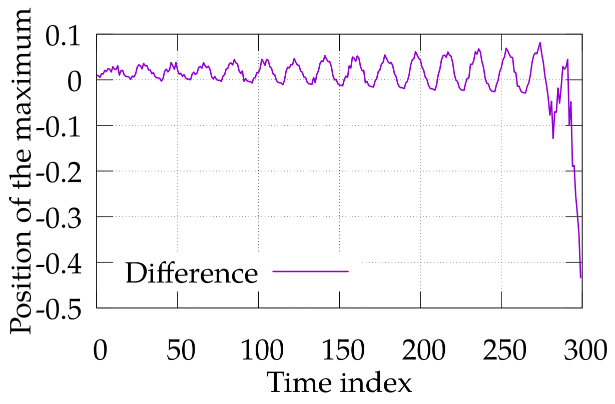

Figure 5: Difference between simulation and ansatz positions of the maximum (fig. 4). The agreement deteriorates as it reaches the border.

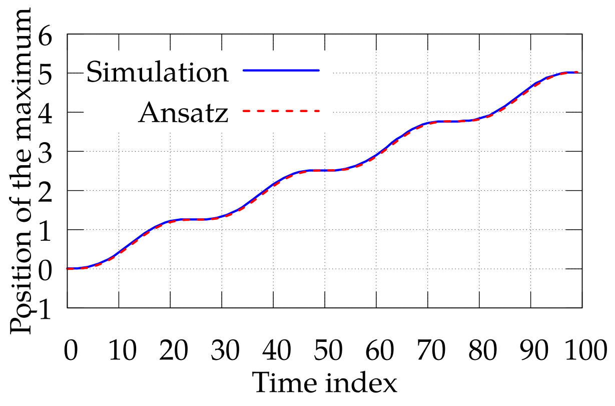

Figure 6: Location of the maximum of the ground state vs. time index. Parameters: $[x_l, x_h] = [-10, 10]$, $\omega = 5$, $T = 5$, $n = 100$, $N = 1024$.

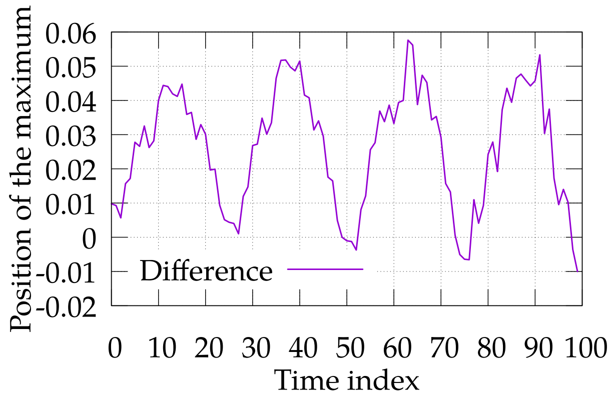

Figure 7: Difference between simulation and ansatz positions of the maximum (fig. 6).

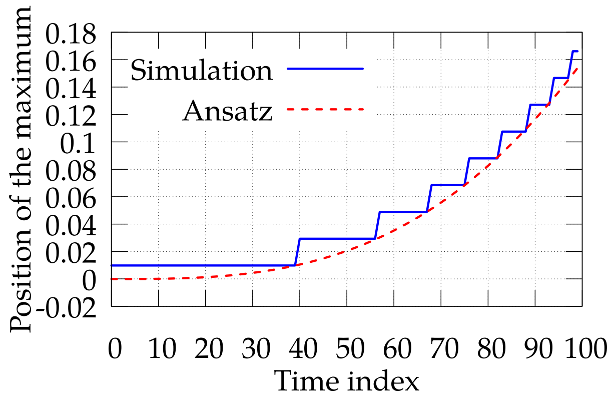

Figure 8: Location of the maximum of the ground state vs. time index. Parameters: $[x_l, x_h] = [-10, 10]$, $\omega = 1$, $T = 1$, $n = 100$, $N = 1024$. The discretization is clearly visible.

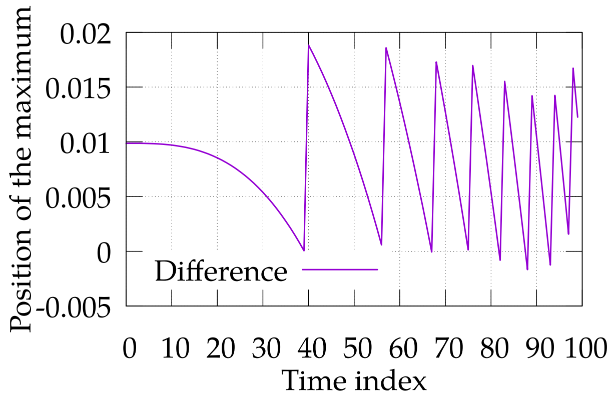

Figure 9: Difference between simulation and ansatz positions of the maximum (fig. 8). The difference is strongly influenced by the discretization.

This sort of composition of the two motions, which resembles classical mechanics, is however not a formal result.

A final remark concerns the shape of the wave: while in general it is not modified, for larger values of T and ω, between the pauses in the motion, the wave noticeably shrinks.

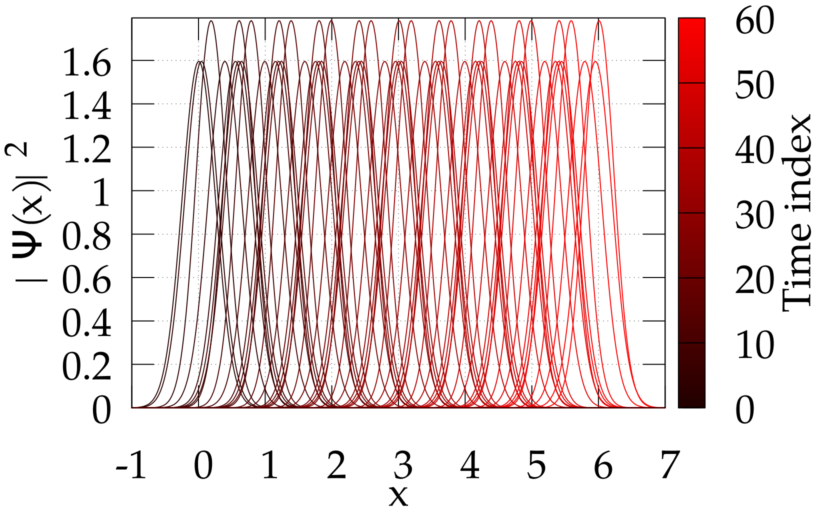

Figure 10: Parameters: $[x_l, x_h] = [-10, 10]$, $\omega = 10$, $T = 10$, $n = 100$, $N = 1024$. Plotted every time step, up to 60, in order to better show the distortion.

This behavior may be caused by the fact that the wave escapes the 'ground energy basin' of the potential (that is, the region of the x axis where the potential is lower than the ground energy) because of both the steepness and the speed of the potential.

Comments and self evaluation

The time-dependent Schrödinger equation was solved using the split operator method. The resulting program was reasonably fast (indeed, the scaling of the DFT, which is the main computational tool of this algorithm, is $\mathcal{O}(n \log n)$), even more so when the number of points in the grid was a power of two (as the DFT algorithms achieve their best timings on arrays of these sizes).

A possible computational improvement could be obtained as follows. After splitting the evolution operator, substitute the first exponential as follows:

$$\exp\left(-\frac{iV(x,t)\Delta t}{2}\right) \longrightarrow \exp\left(-\frac{iV(x,t+\Delta t)\Delta t}{2}\right)$$

Avoiding any discussion about the theoretical soundness of such procedure, one could effectively halve the number of distinct Fourier transforms, as the last exponentiation of a given time step could be joined to the first of the subsequent:

$$e^{-iV(x,t+2\Delta t)/2} e^{-iT(p)\Delta t} e^{-iV(x,t+\Delta t)/2} e^{-iV(x,t+\Delta t)/2} e^{-iT(p)\Delta t} e^{-iV(x,t)/2} \left|\psi(x,t)\right\rangle =$$
$$e^{-iV(x,t+2\Delta t)/2} e^{-iT(p)\Delta t} e^{-iV(x,t+\Delta t)} e^{-iT(p)\Delta t} e^{-iV(x,t)/2} \left|\psi(x,t)\right\rangle ,$$

As a consequence, the algorithm would be more efficient, performing two applications of $V(x,t)$ at once. However, it would not be possible to correctly keep track of the evolution step by step, but only to compute the last value of the state.

As a comment on the assignment, I found that using `FFTW` proved more challenging than it should have, as the documentation is not particularly clear. However, the final result is indeed satisfying.