Exercise 7
Time Dependent Schrödinger Equation

Michele Guadagnini - ID 1230663

November 24, 2020

Abstract

The aim of this exercise is to solve the Schrödinger equation for the time dependent *harmonic oscillator* potential in one dimension and to compute the time evolution of the ground state. The computational method chosen is the *Split Operator Method*.

1 Theory

The hamiltonian of the Schrödinger equation of the harmonic oscillator to be considered can be written as:

$$H = \frac{P^2}{2} + \frac{(Q - Q_0(t)^2)}{2} \quad , \quad where: \quad Q_0(t) = \frac{t}{T} \tag{1}$$

with T the total time of the evolution.

In order to compute the time evolution of the ground state, firstly the ground state wavefunction for $t = 0$ is needed. This can be done by discretizing the space, the second derivative and the potential and then diagonalize the resulting hamiltonian matrix, as done in the Week 6 assignment.

Analitically, the time evolution consists in applying the *time evolution operator* to the wavefunction as:

$$|\Psi(t)>= e^{-i\hat{H}(t)dt}|\Psi(0)> \tag{2}$$

Computationally, it can be done by using the *Split Operator Method*. It consists in separating the kinetic and potential terms of the hamiltonian by accepting an error of the order of dt^2 due to the fact that the two operators do not commute. To reduce this error it is possible to apply the *Strang Splitting*, that consists in separate the potential operator step in two steps of $\frac{dt}{2}$ and allows to reduce the error to the order of dt^3.

Once the kinetic term is separated from the potential one, to speed up computation the *Fast Fourier Transform* (*FFT*) algorithm can be used. It allows to apply the kinetic operator as a diagonal matrix to the transformed wavefunction and then transform it back to space representation. With this trick the overall complexity is reduced from $\mathrm{O}(N^2)$ to $\mathrm{O}(Nlog(N))$.

Summing up, the complete procedure is represented by the following formulas:

$$e^{-i\hat{H}(t)dt} \approx e^{-i\hat{V}\frac{dt}{2}} e^{-i\hat{T}dt} e^{-i\hat{V}\frac{dt}{2}} \tag{3}$$

$$|\Psi(t+dt)>= e^{-i\hat{V}\frac{dt}{2}} \mathcal{F}^{-1} \mathcal{F} e^{-i\hat{T}dt} \mathcal{F}^{-1} \mathcal{F} e^{-i\hat{V}\frac{dt}{2}} |\Psi(t)> \tag{4}$$

The wavefunction is expected simply to shift towards the new minimum of the potential at each time step. This minimum moves with constant speed from $x = 0$ to $x = 1$.

2 Code Development

2.1 Design and Implementation

The implementation started by building up the module *TimeDependentSE* that contains all the subroutines used in this assignment. The most important ones are:

- *InitParamsFromFile*: it reads the four parameters of the problem (LL, TT, $NdivX$, $NdivT$) from the configuration file and stores them in the dedicated derived type variable *Pars*. If something goes wrong while reading, it sets the parameters to default by calling the subroutine *InitDefaults*. It also allows to set the parameters in arbitrary order.
- *MomentumSpace*: it computes the momentum discrete space in the particular way needed to match the output of the *FFT* (see Listing 1).
- *ComplexFFT*: it uses *FFTW3* library to compute the forward or backward Fourier Transform of a complex array. It is built to modify in place the passed array. This part of the code is contained in Listing 1.
- *PrintTimeEvol*: it prints in a file the discrete space and the wavefunction for a particular time step. The files for all the time steps for a single value of the total time T are stored in a dedicated folder.

```
190     SUBROUTINE MomentumSpace(Pars, P_vec, dp)
191         TYPE(Parameters) Pars
192         DOUBLE PRECISION dp
193         DOUBLE PRECISION, DIMENSION(:) :: P_vec
194         INTEGER ii
195         WRITE(*,"(A)") "Computing the momentum space..."
196         dp = 2d0*Pi/Pars%LL
197         !The order of the P_vec elements must match the output of FFT
198         DO ii=1,Pars%NdivX
199             IF (ii .le. (Pars%NdivX/2+1)) THEN
200                 P_vec(ii) = dp*DFLOAT(ii-1)
201             ELSE
202                 P_vec(ii) = dp*DFLOAT(ii -1 - Pars%NdivX)
203             ENDIF
204         ENDDO
205         RETURN
206     END SUBROUTINE
207
208     !It computes the forward Fourier transform of a complex vector
209     SUBROUTINE ComplexFFT(Pars, Cvec, flag)
210         TYPE(Parameters) Pars
211         DOUBLE COMPLEX, DIMENSION(:) :: Cvec
212         DOUBLE COMPLEX, DIMENSION(size(Cvec)) :: temp
213         TYPE(C_PTR) plan
214         INTEGER flag    !-1 for forward transf., +1 for backward
215         CALL dfftw_plan_dft_1d(plan, Pars%NdivX, Cvec, temp, flag,
    FFTW_ESTIMATE)
216         CALL dfftw_execute_dft(plan, Cvec, temp)
217         CALL dfftw_destroy_plan(plan)
218         Cvec = temp
219         RETURN
220     END SUBROUTINE
```

Listing 1: MomentumSpace and ComplexFFT subroutines.

The subroutines regarding the ground state computation at $t = 0$ have been already described in the previous assignment and not mentioned here.

The main program is contained in the file *Ex7-Guadagnini-CODE.f90* and it is divided in three main parts:

- The first part reads the parameters values from the file *Config_ Pars.txt*, initializes the variables and allocates the memory needed for the next part.
- The second one computes the ground state at $t = 0$ of the system and stores it. It also does some tests if debug is active.
- The last one computes the time evolution of the ground state as described in Section 1. It is mainly constituted by a *DO* loop over the time steps, in which the appropriate subroutines are called. This part is reported in Listing 2.

```
!-- COMPUTING THE TIME EVOLUTION ---------------------------------------

    WRITE(*,"(A)") "Starting to compute the time evolution of the ground
    state ..."

    ALLOCATE(P_vec(Pars%NdivX))
    CALL MomentumSpace(Pars, P_vec, dp)
    CALL Checkpoint(Pars%Debug, "Momentum space computed successfully.")

    Psi_t = DCMPLX(Psi_0, 0d0)
    CALL PrintTimeEvol(output, 0, Psi_t, X_vec, TT_str)

    WRITE(*,"(A)") "Starting the Split Operator Method computations ..."

    DO tstep=2, size(T_vec)
        !Update the potential array to the new time step
        CALL HarmonicOscillatorPot(Pars, X_vec, v_pot, T_vec(tstep) )

        ! 1) half potential operation
        Psi_t = Psi_t * CDEXP( DCMPLX(0d0, -v_pot*(dt*0.5d0)) )

        ! 2) Fourier transform of Psi_t
        CALL ComplexFFT(Pars, Psi_t, -1)

        ! 3) kinetic term operation in momentum space
        Psi_t = Psi_t * CDEXP( DCMPLX(0d0, -0.5d0*(P_vec**2)*dt) )

        ! 4) Inverse Fourier transform of Psi_t
        CALL ComplexFFT(Pars, Psi_t, +1)
        Psi_t = Psi_t/DCMPLX(Pars%NdivX, 0d0) !normalization

        ! 5) second half potential operation
        Psi_t = Psi_t * CDEXP( DCMPLX(0d0, -v_pot*(dt*0.5d0)) )

        !Saving the computed time evolution in a file for each time step
        IF ( (MOD(tstep,100) .eq. 0) .or. (tstep .eq. size(T_vec)) ) THEN
            CALL PrintTimeEvol(output, tstep, Psi_t, X_vec, TT_str)
            IF (Pars%Debug) THEN
                CALL PrintPotEvol(output, tstep, v_pot, X_vec)
            ENDIF
        ENDIF
    ENDDO
```

Listing 2: Time evolution computation code.

2.2 Debug and Test

The *FFTW3* library has been installed with the command:

```
sudo apt install libfftw3-dev libfttw3-3
```

Then the program has been compiled and executed with:

```
gfortran *CODE.f90 -o TimeDepSE.x -lblas -llapack -lfftw3 -lm -g -fcheck=
    all -Wall -Wconversion-extra
./TimeDepSE.x
```

To debug the program some additional subroutines has been used together with the usual *Debugging* module. Also some printings has been implemented in the main program, that allows to test the correctness of intermediate results, such as the potential computation, the eigenpairs resulting from the diagonalization, the output of the Fourier Transform, etc.

During the testing of the complete program, it happened that the wavefunction for different time steps was not moving towards the new minimum, remaining completely unchanged. The problem was that the subroutine *dfftw_plan_dft_1d* from *FFTW3* was called by passing as fifth argument the string $FFTW\_FORWARD$ or $FFTW\_BACKWARD$ instead of integer numbers -1 and $+1$. The subroutine was working fine anyway at the first time step, but in the following steps probably some memory issues happened. Indeed, the output in the test file *FFTtest.txt* was correct also with the string arguments, making this error difficult to figure out.

3 Results

The values of the parameter TT that has been tested are: 1, 5, 10, 50, 100, 500. The other parameters have been kept fixed to the following values: $NdivX = NdivT = 1024$ and $LL = 15$.

The results have been plotted with *GNUPLOT* and they are reported in Figure 1. It is possible to notice that for small values of the parameter TT, in particular for $TT = 5$ and lower (Figure 1a and 1b), the maximum of the wavefunction can overcome or not reach at all the $x = 1$ position. Also, in the same plots it seems that the wavefunction is not moving with constant speed. This could be due to the time step that starts to be too small.

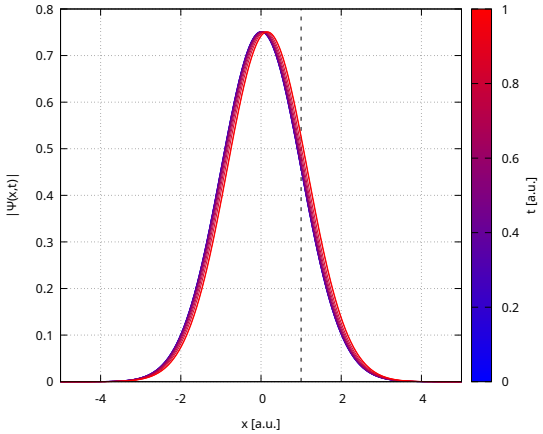
For values of TT bigger than 100 the maximum value of the wavefunction starts to oscillate, meaning that the time step starts to be too big to have good precision in computation of *FFT* (Figure 1f).

4 Self-evaluation

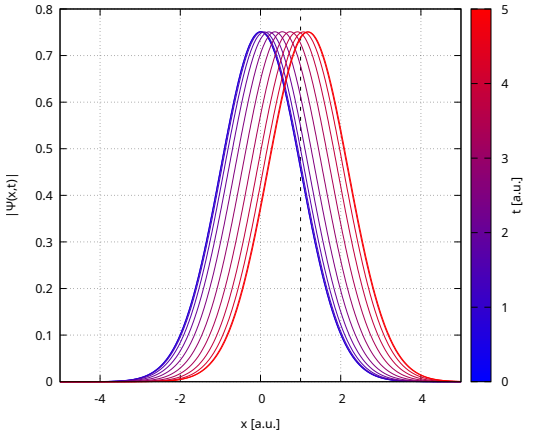
Things learned while completing this assignment are:

- to apply the Split Operator Method to solve a Time Dependent Schrödinger equation;
- to use the Fast Fourier Transform on an array.

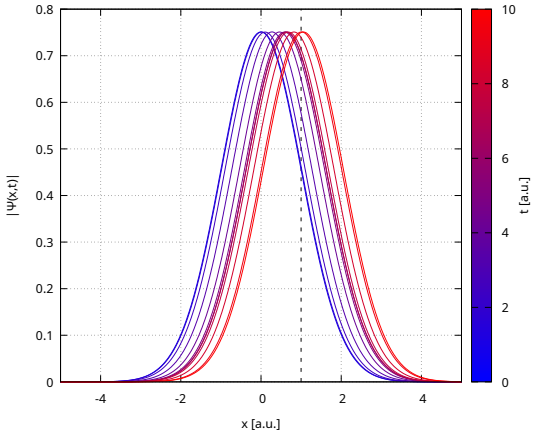
To improve the program it would be better to separate ground state computation from the time evolution. This would allow to increase the flexibility of the code. It would have been interesting to calculate the peak position for each time step and see how it moves. It seems not to move with constant speed but with oscillating speed. Another possible improvement could be to allow comments in configuration file.



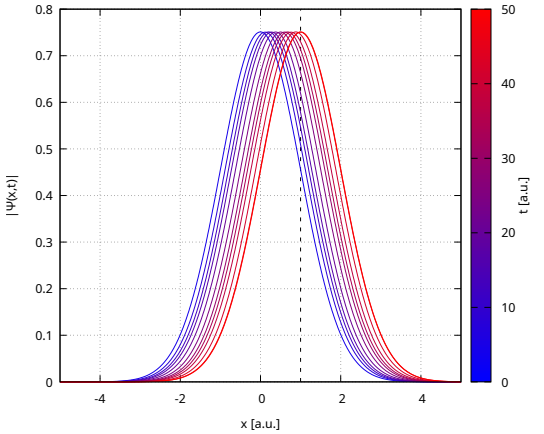
(a) Time evolution with parameter $TT = 1$.



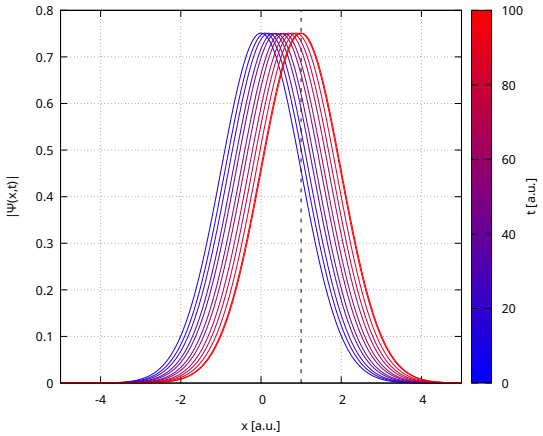
(b) Time evolution with parameter $TT = 5$.



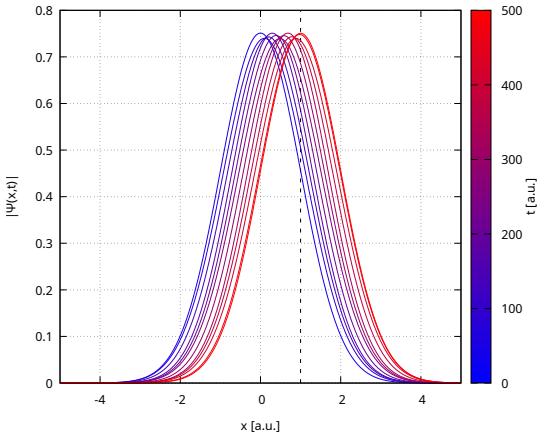
(c) Time evolution with parameter $TT = 10$.



(d) Time evolution with parameter $TT = 50$.



(e) Time evolution with parameter $TT = 100$.



(f) Time evolution with parameter $TT = 500$.

Figure 1: Time evolution for different values of the parameter TT. The plotted curves are taken every 100 time steps.