SOLUTION OF TIME DEPENDENT SCHRÖDINGER EQUATION - STUDY OF QUANTUM WAVEPACKET MOTION

Alessandro Cuoghi, Matteo Quinzi, Giacomo Rizzi

Unimore - Laboratorio di Fisica Computazionale

Summary: *The time dependent Schrödinger equation is solved numerically using a split-step Fourier algorithm. It is considered the motion of a quantum particle in a region of space in presence of a potential energy: it is studied the case in which the wavepacket is scattered by a single (or a double) square potential barrier and the motion inside a potential well. The wavefunction of the particle is described by a Gaussian wavepacket.*

1. Introduction

When it comes to study the motion of a particle in a space of atomic size, for example an electron moving in a region in presence of a potential energy, the problem has to be treated quantum mechanically. The behaviour of the particle is found by solving the **time dependent Schrödinger equation**, which describes its evolution through time. Considering a particle with a defined initial momentum and a localized position, the **uncertainty principle** implies that the solution is given by a wavepacket; it also ensures that the wavepacket broadens during the motion, depending the dispersion relation on the precision over the initial localization of the particle, i.e. on the spatial width of the wavepacket.

2. System definition and algorithms description

The one dimensional time dependent Schrödinger equation is

$$i\hbar\frac{\partial}{\partial t}\psi(x,t) = \mathcal{H}\psi(x,t), \tag{1}$$

where $\mathcal{H}$ is the **Hamiltonian** of the system:

$$\mathcal{H} \equiv -\frac{\hbar^2}{2m}\frac{\partial^2}{\partial x^2} + \mathcal{V}(x). \tag{2}$$

The first term is the **kinetic energy operator** $\mathcal{T}$, while $\mathcal{V}$ represents the **potential energy operator**. In coordinate space $\mathcal{T}$ is thus a differential operator, while $\mathcal{V}$ is multiplicative. Fourier transforming to momentum space, the Schrödinger equation results

$$i\hbar\frac{\partial}{\partial t}\tilde{\psi}(p,t) = \frac{p^2}{2m}\tilde{\psi}(p,t) + \frac{1}{\sqrt{2\pi\hbar}}\int_{-\infty}^{\infty} dq\tilde{V}(p-q)\tilde{\psi}(q,t), \tag{3}$$

showing that the kinetic energy operator becomes a multiplicative operator. This discrimination between the mathematical forms of the operators it is the fundament of the algorithm developed for the numerical computation.

Given the initial wavefunction $\psi(x,0)$, in addition to boundary conditions, the equation (1) has a unique solution of the form:

$$\psi(x,t) = e^{-\frac{i}{\hbar}\mathcal{H}t}\psi(x,0). \tag{4}$$

Since $\mathcal{T}$ and $\mathcal{V}$ do not commute, the evaluation of the exponential operator, the **evolution operator**, is very complicated. It is possible to manipulate this operator by considering a small timestep τ, so that it becomes:

$$e^{-\frac{i}{\hbar}\mathcal{H}t} = e^{-\frac{i}{\hbar}(\mathcal{T}+\mathcal{V})\tau} \approx e^{-\frac{i}{\hbar}\mathcal{T}\tau}e^{-\frac{i}{\hbar}\mathcal{V}\tau}. \tag{5}$$

This approximation is called the **Suzuki-Trotter expansion** and has an error of $\mathcal{O}(\tau^2)$; the last term represents the **approximate propagator**. The symmetric factorization

$$e^{-\frac{i}{\hbar}(\mathcal{T}+\mathcal{V})\tau} \approx e^{-\frac{i}{2\hbar}\mathcal{V}\tau}e^{-\frac{i}{\hbar}\mathcal{T}\tau}e^{-\frac{i}{2\hbar}\mathcal{V}\tau} \tag{6}$$

yields to an error of $\mathcal{O}(\tau^3)$ in addition to being a **unitary operator**, implying that the probability density is conserved throughout the computation.

2.1. The algorithm

The In showing the algorithm designed, the notation is simplified by setting $m = 1$ and $\hbar = 1$.

Noticing that in coordinate space the potential operator $\mathcal{V}$ is multiplicative while $\mathcal{T}$ is a differential operator, and vice versa, in momentum space, the latter becomes a multiplicative operator, while the other not, it is possible to find an approximation to the Schrödinger equation solution by using an algorithm that relies on the factorization (6) combined with **Fourier transform**. Given the wave function at a certain time t, its evolution after a small time step τ is computed in three steps:

- Starting in coordinate space, the wave function $\psi(x,t)$ is multiplied by the first half step potential evolution operator:

$$\psi'(x) = e^{-\frac{i}{2}\mathcal{V}\tau}\psi(x,t). \tag{7}$$

- Through a Fourier transform, the wave function is computed in momentum space. Then, it is multiplied by the kinetic evolution operator, which has become diagonal:

$$\tilde{\psi}'(p) = \frac{1}{\sqrt{2\pi}}\int_{-\infty}^{\infty} dx\ e^{-ipx}\psi'(x), \tag{8}$$

$$\tilde{\psi}''(p) = e^{-\frac{i}{2}p^2\tau}\tilde{\psi}'(p). \tag{9}$$

- The final wave function is obtained by Fourier transforming back to position space and multiplying for the half step potential evolution operator:

$$\psi''(x) = \frac{1}{\sqrt{2\pi}}\int_{-\infty}^{\infty} dp\ e^{ipx}\tilde{\psi}''(p), \tag{10}$$

$$\psi(x,t+\tau) = e^{-\frac{i}{2}\mathcal{V}\tau}\psi''(x). \tag{11}$$

This simple and efficient algorithm allows to find at each run an approximated solution to Schrödinger equation and, thus, to evaluate the time evolution of the wavepacket. This decomposition of the time interval in small timesteps is suitable for a numerical approach to the problem. In addition, the extensive use of the Fourier transform improves the performance of the code developed, since there exists optimized methods to compute the transform, known as Fast Fourier Transform algorithm.

In order to implement a numerical computation, a finite spatial interval $[0, L]$ is chosen, discretized in N-inner points. The potential energy and the wavefunction are defined over this interval, thus:

$$V \equiv V_j = V(x_j), \tag{12}$$

$$\psi \equiv \psi_j = \psi(x_j). \tag{13}$$

The initial wave packet consists in a normalized Gaussian function multiplying a plane wave:

$$\psi(x,0) = \left(\frac{1}{2\pi\sigma^2}\right)^{\frac{1}{4}} e^{-\frac{(x-x_0)^2}{4\sigma^2}} e^{ik_0 x}. \tag{14}$$

The position of the wavepacket is determined by x_0, while k_0 sets the initial momentum; σ defines the width of the wavepacket, i.e. its space localization. This parameter entails the **wavepacket spreading** during the motion, caused by the uncertainty principle: the momentum uncertainty is $\Delta k = 1/2\sigma$ leading to a precision over the future position by the amount of $t/2\sigma$ and thus the shape of the wavepacket broadens with the time passing.

3. The developed code

The code, written in **FORTRAN**, computes the time evolution of the wavepacket by numerical solving Schrödinger equation (1) using the algorithm reported in (2.1). The code consists in a main program linked with two modules: the module `READ_VAR`, used to read from an external file the values of the quantities employed and `TASK`, that contains the routines called by the head program `MAIN` in order to perform the calculus.

3.1. READ_VAR

```
1  MODULE READ_VAR
     ! Declares the initial variables.
3    ! Some of them are read from an external file.
     IMPLICIT NONE
5
     ! Global values accessible from the other programs:
7    ! constant values and routine parameters
     REAL(KIND=8) :: pi = 4.D0 * DATAN(1.D0)   ! pi
9    COMPLEX(KIND=8) :: im = (0.D0, 1.D0)   ! imaginary unit
     INTEGER :: i, save_timestep, IOS
11
     ! values to be read from an external file
13   INTEGER :: N, M, save_wave
     REAL(KIND=8) :: L, E_1, l_1, E_2, l_2
15   LOGICAL :: load_f_inp
     CHARACTER(50) :: f_inp
17   REAL(KIND=8) :: sigma, x0, k0, dt
```

Listing 1: `READ_VAR` module. Contains a single subroutine that reads the values of the parameters required by the code from an external file.

The first part of the module is made of variables declarations accessible from all the other programs. They consist of some constant (such π and i), routine parameters and the variables characterizing the wavepacket and its time evolution ; the latter are read from an external file, in order to give the program a more usability and a faster setup (otherwise the code has to be compiled every time a value is changed). The only subroutine in the module is called `READ_DATA`. It first opens the file `f_data`, stopping the program and returning an error message if the file does not exist or is not readable, and then it reads the file line by line. The parameters that are read are presented and discussed in the following sections.

```
SUBROUTINE READ_DATA()
  ! Read values from an external file.
  IMPLICIT NONE
  CHARACTER(20) :: f_data = 'Input/data_sheet.dat'

  PRINT *, 'Reading values from "', f_data, '"...'
  OPEN(UNIT=10, FILE=f_data, STATUS='OLD', ACTION='READ', IOSTAT=IOS)
  IF (IOS==0) THEN
    READ(10, *)
    READ(10, '(3X, I10)') N ! number of points of the interval
    READ(10, '(3X, F20.10)') L ! length of the interval
    [...]
  ELSE
    PRINT *, 'Error in opening "', f_data, '". Program stopping...'
    STOP
```

```
## INTERVAL [0, L]
N= 10000                    number of points
L= 90.                      length of the interval
```

Listing 2: A the top, subroutine `READ_DATA`: the code opens the input file *data_sheet.dat*, located in the *Input* directory, and reads the values of the corresponding variables. The listing shows only the reading of the first three rows. At the bottom, the first three lines of the input file *data_sheet.dat*.

3.2. TASK

```
MODULE TASK
  ! Defines the routines called by the main program to perform the computation.
  USE READ_VAR    ! Gains access to the variables declared in READ_VAR module.
```

Listing 3: `TASK` module. Contains the subroutines called by the main program in order to perform the computation.

The module `TASK` contains three functions used by the main program to compute the time evolution of the wavepacket. The first function `POT` designs the shape of the potential energy by building two barriers centered, respectively, at $L/2$ (half-interval) and $2L/3$ (two thirds of the interval). Together with the length of the interval `L` and the number of points `N`, the length `l_1` and `l_2` and the height `E_1` and `E_2` of the barriers are read from the input file. The parameters `l1` and `l2` represents the semilengths of the barriers, in array units, in order to represent subsets of the space interval.

```
FUNCTION POT()
  ! Builds the potential energy array.
  IMPLICIT NONE

  ! OUTPUT
  REAL(KIND=8) :: POT(N)    ! potential energy array
```

```
7
   ! ROUTINE
9  INTEGER(KIND=8) :: l1, l2    ! barriers widths

11 l1 = INT(N * l_1 / (2 * L))    ! semilentgh of the first barrier in array points
       unit
   l2 = INT(N * l_2 / (2 * L))
13
   ! The potential energy is everywhere zero except for two regions where it has a
       constant values of E_1 and E_2.
15 POT(:) = 0.D0
   POT(  N/2 - l1 :   N/2 + l1) = E_1    ! first barrier
17 POT(2*N/3 - l2 : 2*N/3 + l2) = E_2    ! second barrier
END FUNCTION POT
```

Listing 4: `POT` function. The two barriers are centered at half and two-third of the interval, their dimensions are read from the input file.

The task of the function `PHI` consists in building the initial wavepacket, by returning a N-points array containing the values of the wavefunction computed over the spatial interval array `x`.

```
FUNCTION PHI(x)
2  ! Returns the initial wavepacket array, computed over the points of the spatial
       interval array x.
   IMPLICIT NONE
4
   ! INPUT
6  REAL(KIND=8), INTENT(IN) :: x(N)    ! spatial interval array

8  ! OUTPUT
   COMPLEX(KIND=8) :: PHI(N)    ! wavepacket array
```

Listing 5: `PHI` function. Returns the initial wavepacket array `PHI` computed over the x points of the interval.

The behaviour of the routine depends on the value of the parameter `load_f_inp`, read from the input file: if `TRUE`, the wavefunction is read from a external file specified by the variable `f_inp`; a check on the reading instructions ensures that the file exists and is readable, stopping the execution of the program if not. If `FALSE`, the function returns the Gaussian wavepacket (14).

```
1 IF (load_f_inp .eqv. .TRUE.) THEN
    OPEN(UNIT = 10, FILE = f_inp, STATUS = 'OLD', ACTION = 'READ', IOSTAT = IOS)
3   IF (IOS == 0) THEN
      READ(10, '(2F20.10)') (PHI(i), i = 1,N)
5   ELSE
      PRINT *, 'Error in opening "', f_inp, '". Program stopping...'
7     STOP
    ENDIF
9 ELSE
    PHI(:) = (2.D0 * pi * sigma**2)**(-0.25) * EXP(im * k0 * x(:)) * EXP(-(x(:) -
      x0)**2 / (4.D0 * sigma**2))
```

Listing 6: The core of the function consists of an `IF` statement: the wavepacket is either read from an input file or it is shaped as a Gaussian wavepacket.

The last routine `FFT` computes the discrete Fourier transform of an input array `PHI` by

calling the subroutines of the external library **FFTW** ([1]). At the beginning of the function, the statement `INCLUDE` links the program with the library.

```
FUNCTION FFT(PHI, DIR)
  ! Computes the discrete Fourier transform of an input array PHI. The
    computation is performed with the help
  ! of the external subroutine library FFTW (http://www.fftw.org).
  IMPLICIT NONE
  INCLUDE 'fftw3.f'  ! FFTW interface

  ! INPUT
  COMPLEX(KIND=8), INTENT(IN) :: PHI(N)  ! input array to transform
  CHARACTER(LEN=1), INTENT(IN) :: DIR   ! transform direction

  ! OUTPUT
  COMPLEX(KIND=8) :: FFT(N)   ! tranformed array

  ! ROUTINE
  INTEGER(KIND=8) :: plan   ! integer storing the information about the transform
```

Listing 7: `FFT` function. Computes the Discrete Fourier Transform using the routines contained in FFTW library.

The direction of the Fourier transform is controlled by the input parameter `DIR`: '`F`' stands for "forward" where the exponent in the Fourier transform takes the negative sign, while '`B`' is for "backward" as the exponent is positive. `plan` is an integer storing the information about the transform: the computation is performed by executing this plan, as it is showed in listing (8).

```
  IF (DIR == 'F') THEN
    CALL DFFTW_PLAN_DFT_1D(plan, N, PHI, FFT, FFTW_FORWARD, FFTW_ESTIMATE)   !
    forward transform
  ELSEIF (DIR == 'B') THEN
    CALL DFFTW_PLAN_DFT_1D(plan, N, PHI, FFT, FFTW_BACKWARD, FFTW_ESTIMATE)   !
    backward transform
  ENDIF

  CALL DFFTW_EXECUTE_DFT(plan, PHI, FFT)   ! subroutine call to execute the
    calculus
  CALL DFFTW_DESTROY_PLAN(plan)
```

Listing 8: The FFTW library computes the Fourier transform in few steps. First, a plan is created by pointing the name of the initial and final arrays as well of the transform direction; then the plan is executed and the final array *FFT* is returned.

The flag `FFTW_ESTIMATE` "specifies that, instead of actual measurements of different algorithms, a simple heuristic is used to pick a (probably sub-optimal) plan quickly" ([2]). Eventually, the function destroys the plan and returns the computed array.

3.3. MAIN

```
PROGRAM MAIN
```

([1]) From `http://www.fftw.org` - "FFTW is a C subroutine library for computing the discrete Fourier transform (DFT) in one or more dimensions, of arbitrary input size, and of both real and complex data."

([2]) from *FFTW* manual document.

```
!The program numerically solves the time-dependent Schroedinger equation by
  using a split-step algorithm.
USE READ_VAR  ! Gains access to the variables declared in READ_VAR module.
USE TASK    ! TASK module contains the functions used to perform the calculus.

IMPLICIT NONE

INTEGER :: l2, l1    ! barriers semilengths converted in array points unit

! Array declarations:
REAL(KIND=8), ALLOCATABLE :: x(:), V(:), k(:), PHI_EVOL(:,:), PHI_SQMOD(:)
COMPLEX(KIND=8), ALLOCATABLE :: V_OP(:), K_OP(:), PHI_X(:), PHI_K(:)
```

Listing 9: The `MAIN` program. Contains the code to perform the computation.

The motion of the wavepacket is computed by the `MAIN` program. The head of the code contains the `USE` statement, to call the modules `READ_VAR` and `TASK`, and the declaration of the allocatable arrays. After printing some information about the program, by calling the subroutine `TITLE`, the subroutine `READ_DATA` is called and the variables from the input file are read; thus, the arrays are allocated.

```
CALL TITLE()    ! message with some information about the code
CALL READ_DATA()    ! Reads the variables from the external file.
! Array allocations:
ALLOCATE(x(N), V(N), k(N))
ALLOCATE(V_OP(N), K_OP(N))
ALLOCATE(PHI_X(N), PHI_K(N), PHI_SQMOD(N), PHI_EVOL(N, save_wave+1))

x(:) = (/ (i * L / (N - 1), i = 0, N-1) /)    ! spatial interval array

k( : N/2) = (/ (2.D0 * pi * i / L, i = 0, N/2-1) /)
k(N/2 + 1 : ) = (/ (-2.D0 * pi / L * (N/2.D0 - i), i=0, N/2-1) /)
K_OP(:) = EXP(-im * k(:)**2 * dt / 2.D0)    ! kynetic time evolution operator
  array

V = POT()    ! potential energy array
V_OP(:) = EXP(-im * V(:) * dt / 2.D0)    ! potential time evolution operator

l1 = INT(N * l_1 / (2 * L))    ! semilength of the first barrier in array points
   unit
l2 = INT(N * l_2 / (2 * L))

PHI_X = PHI(x)    ! initial wavepacket

PHI_EVOL(:,1) = REAL(PHI_X(:))**2 + AIMAG(PHI_X(:))**2
```

Listing 10: The initial part of the program: the code first prints some information about the program and the authors, calls `READ_DATA` (3.1) to read the input file and allocates the arrays; the discrete interval x and the wavevectors k arrays are then computed, together with the potential energy array `V` and the wavepacket `PHI_X`, by calling the corresponding routines contained in `TASK` module (3.2.

The x array contains the N points of the $[0, L]$ interval. k represents the points of the momentum space, i.e. the wavevectors; reflecting the output ordering of the computed discrete Fourier transform, the array is build in such a way that in the first half the positive wavevectors are stored in ascending order, while the negatives are stored in the second half, still in ascending order. The function `POT` then returns the potential energy array and the two time evolution operators defined in (6) are computed. $l1$ and $l2$ are the semilengths of the two barriers converted in array points unit. The array `PHI_X` contains the values of the wavepacket and it is built by calling the subroutine `PHI`. The first wavepacket composes the first column of the array `PHI_EVOL`. The core of the program is the `DO` cycle showed in listing (11); the code relies on the three-steps

algorithm defined in (2.1) where the time evolution is decomposed using factorization (6), Fourier transforming backward and forward permits to switch from coordinate space to momentum space and conversely. The time interval considered is $M \cdot dt$, where `M` is the total number of timesteps while `dt` is the timestep width.

```
save_timestep = M / save_wave    ! time evolution storing frequency

DO i = 1, M
  PHI_X(:) = PHI_X(:) * V_OP(:)    ! half-timestep time evolution computed in
    coordinate space

  PHI_K = FFT(PHI_X, 'F')    ! Fourier transform ('forward') to momentum space
  PHI_K(:) = PHI_K(:) * K_OP(:) / N    ! time evolution in momentum space

  PHI_X = FFT(PHI_K, 'B')    ! Fourier antitransform ('backward') back to
    coordinate space
  PHI_X(:) = PHI_X(:) * V_OP(:)    ! second half-timestep time evolution

  PHI_SQMOD(:) = REAL(PHI_X(:))**2 + AIMAG(PHI_X(:))**2    ! wavepacket square
    modulus
  IF (MOD(i, save_timestep) == 0) PHI_EVOL(:, i / save_timestep + 1) = PHI_SQMOD
    (:)
END DO
```

Listing 11: The core of the `MAIN` program: the first lines inside the `DO` cycle reflect the steps of the algorithm (2.1).

At certain runs of the cycle, if a control condition is matched, the square modulus of the wavepacket is stored in the array `PHI_EVOL`: the storing frequency is defined by the parameter `save_timestep`, calculated in such a way that the array contains `save_wave` + 1 wavepacket corresponding to equispaced time instants. In the last part of the program the subroutine `WRITE_INTO_FILE` is called: first the potential energy array is written together with the spatial interval in a text file `f_pot`; then, the time evolution of the wavepacket is written into `f_evol` file, each column corresponding to a wavepacket of a certain time instant, and the final wave calculated is stored in `f_last`. To resume the last computed evolution, it is possible to point this as the source file to intialize the wavepacket.

```
CALL WRITE_INTO_FILE()    ! writing the computed values in external text files

PRINT *, 'The program has successfully completed the computation.'
PRINT *, 'Good bye.'
PRINT *, ''
PRINT *, ''
```

Listing 12: The final part of the program: the subroutine `WRITE_INTO_FILE` is called and two printings inform the user that the computation has finished.

Eventually the computed parameters of the time evolution are appended into the file `f_coef`: `T` and `R` are, respectively, the transmission and reflection coefficient while `A1`, `A12` and `A2` are the absorption coefficients of the two barriers and the region between them. All these coefficients are computed as the norm of the wavefunction of the corresponding interval. Then the initial and the final norm computed over the entire space are written to test the accuracy of the calculus. Since these values are appended at each run, at the beginning of each printing the input parameters are written in order to associate the computed values with the corresponding simulation.

Eventually, a "good bye" message tells the user that the program has completed the computation and stopped its execution.

```
SUBROUTINE WRITE_INTO_FILE()
  CHARACTER(50) :: FMT
  CHARACTER(14) :: f_pot  = 'Output/pot.txt'
  CHARACTER(17) :: f_evol = 'Output/moving.txt'
  CHARACTER(21) :: f_last = 'Output/final_wave.txt'
  CHARACTER(16) :: f_coef = 'Output/coeff.txt'

  PRINT *, 'Writing potential energy in "', f_pot, '"...'
  OPEN(UNIT=10, FILE=f_pot)
    WRITE(10, '(2F20.10)') (x(i), V(i), i = 1, N)
   [...]
  PRINT *, 'Saving wavepacket evolution in "', f_evol, '"...'
  WRITE(FMT, '("(", I0, "F20.10)")') save_wave + 1
  OPEN(UNIT=10, FILE=f_evol)
    WRITE(10, FMT) (PHI_EVOL(i,:), i=1,N)
   [...]
  OPEN(UNIT=10, FILE=f_coef, POSITION='APPEND')
    WRITE(10, '("L=", F20.10)') L
    WRITE(10, '("E1=", F20.10, ", l1=", F20.10, ", E2=", F20.10, ", l2=", F20.10)
    ') E_1, l_1, E_2, l_2
    [...]
    ! reflection coefficient: wavepacket norm before the first barrier
    WRITE(10, '(A, F15.10, A)') 'R  = ', SUM(PHI_EVOL(: N/2-l1, save_wave+1)) * (
    x(2)-x(1)) * 100.D0, ' %'
    ! first barrier absorption coefficient: wavepacket norm inside the first
    barrier
    WRITE(10, '(A, F15.10, A)') 'A1 = ', SUM(PHI_EVOL(N/2-l1 : N/2+l1, save_wave
    +1)) * (x(2)-x(1)) * 100.D0, ' %'
```

Listing 13: Eventually the code writes the calculated quantities into external text files.

```
L=        90.0000000000, N=      10000
E1=         0.0000000000, l1=         0.0000000000, E2=         1.0000000000, l2=
          60.0000000000
s=          0.5000000000, x0=        15.0000000000, k0=         0.0000000000
M=     100000, dt=          0.0010000000

R  =    90.1111687064 %
A1 =     0.0019455769 %
A12=     0.0000000000 %
A2 =    12.4416683014 %
T  =     0.0205577770 %

Initial norm = 1.0000000000
Final norm = 1.0000000000
```

Listing 14: The final coefficients and the norm of the wavepacket are stored in a text file, appending each time to the file.

4. Numerical experiments

In this section, the motion of a Gaussian wavepacket $\psi(x,t)$ is studied mainly through numerical experiment of scattering from single and double potential barriers. The wavefunction describes a quantum particle of mass $m = 1$ moving with energy $\frac{k_0^2}{2}$; its square modulus $|\psi(x,t)|^2$ represents the probability density to find the particle in a certain position x at time t and the sum of the probability densities over a certain spatial region is thus the probability to find the particle in that specific region. The computation is performed over the discrete interval $[0, L]$ and the time evolution is calculated through subsequent timesteps dt.

The computation parameters and the results are presented in **atomic units**: the length unit is the **Bohr radius** ($a_0 = 5.291772 \cdot 10^{-11} m$), energies are expressed in **Hartree** ($E_h = 4.359745 \cdot 10^{-18} J$) and the **time unit** equals $2.418884 \cdot 10^{-17} s$. In the following experiments, the interval $[0, 90]$ is chosen, discretized in 10000 points; the timesteps width is set to 0.001.

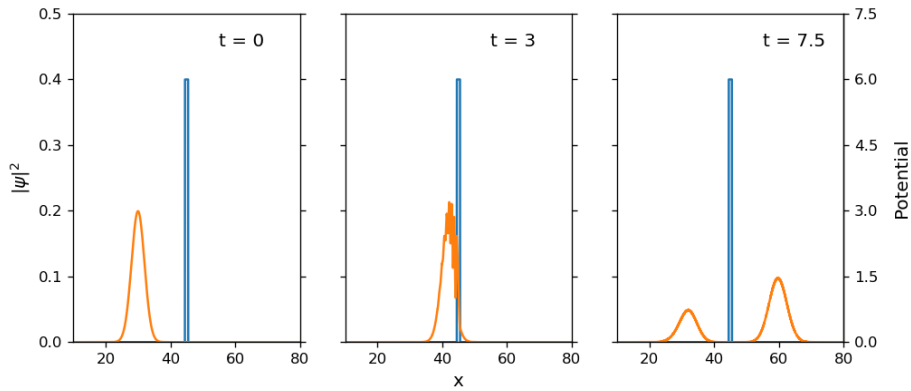

Fig. 1: The scattering of a wavepacket from a potential barrier. The probability density $|\psi(x,t)|^2$ is shown as a function of the coordinate x at three different times. On the right axis, it is reported the potential energy scale.

In figure (1) it is shown the computed time evolution of the wavepacket in presence of a single potential barrier. Two waves moving in opposite direction originate from the scattering as the energy of the wavepacket is slightly greater than the barrier. The scattering experiments examine the coefficients of reflection and transmission, computed as the norm of the corresponding waves divided for the norm of the incident wavepacket; thus, it's crucial to have the probability density conserved throughout the calculus. Figure (2) shows the reflection, absorption and transmission coefficients, relative to the scattering reported in figure (1), computed at each timestep, together with the norm of the wavefunction. The latter is constant to a high level of precision, proving

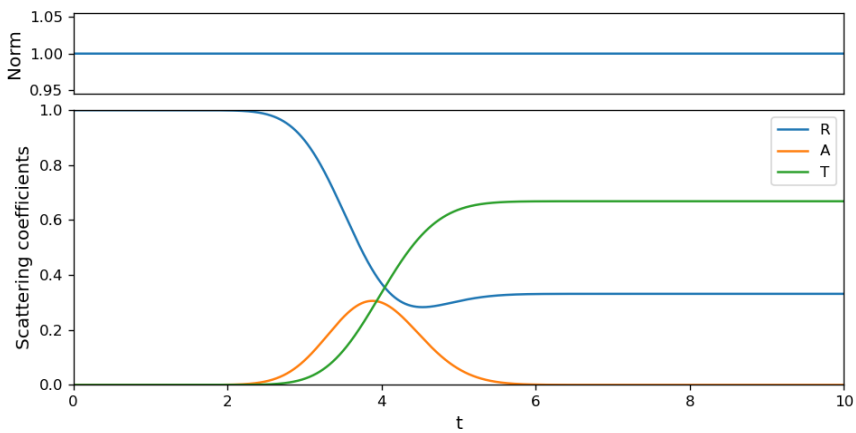

Fig. 2: At the top, the norm of the wavefunction calculated over the entire interval, constant for the whole duration of the evolution. At the bottom, the scattering coefficients of the example reported in figure (1). The wavepacket, which has an initial momentum $k0 = 4$ and a width $\sigma = 2.$, scatters from a single potential barrier tall 6 and large 1. The labels R, A, T represent the reflection, absorption and transmission coefficients as a function of time.

the fact that the time evolution operator factorization is a unitary operator and preserves the norm of the wavefunction. Since the incident wave is normalized, the coefficients are calculated summing the probability density over the points of the corresponding interval; therefore they represents the probability to find the wavepacket in a specific region of space: T is computed as the norm of the wavepacket calculated over the points after the barrier, R before, and A inside the barrier. By studying the coefficients it is possible to reconstruct the path of the wavepacket: first located before the barrier, it enters the potential region generating the reflected and the transmitted wave.

Periodic boundary conditions are imposed by using the DFT routines to compute Fourier transform; it means that the wavefunction escaping from the right limit at $x = L$ reappears at $x = 0$. The space is thus a monodimensional torus. It is possible to take advantage of this

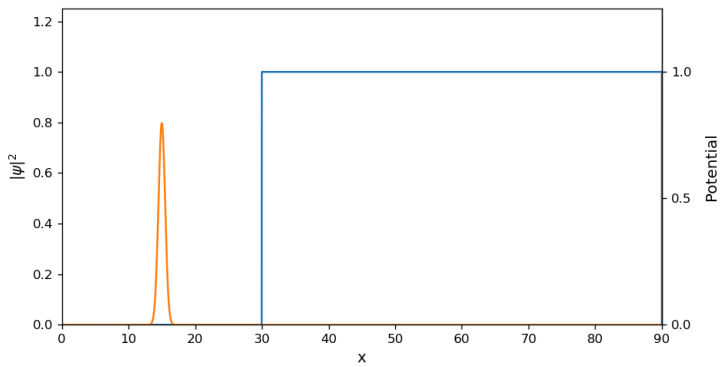

Fig. 3: By extending the right wall of the barrier to the endpoint of the interval, a potential well is obtained at the beginning of the interval. This is a consequence of the periodic boundary conditions as the points $x = 0$ and $x = L$ coincide.

fact and building a potential well: placing the potential barrier in $x = 2L/3$, in such a way that its right wall coincides with the upper limit of the interval $x = L$, corresponds to building a potential well at the beginning of the interval of width $L/3$. This construction is graphed in figure (3) for a better understanding. Inside the well, the shape of the wavepacket is first considered: the parameter σ controls the width of the function i.e. the space localization. The uncertainty principle sets a limit on the precision about the value of the linear momentum thus depending on the precision about the space position. At $t = 0$ a narrow wavepacket is located in the potential well with zero momentum; as subsequent timesteps are considered the wavepacket flattens due to momentum uncertainty and exists a non null probability to find it outside the potential well. The situation is shown in figure (4) where the probability to find the particle in the well does not remain constant at 1, but decreases as the wavepacket exits the well becoming stable around a value of 0.85. Then, it is studied the behaviour of a wavepacket with positive momentum inside the well. Since the scattering phenomenon depends on the energy of the particle and not on its shape, as the concern is only about the particle being inside or outside the well, a wavepacket with a better momentum definition is considered by increasing the width of the Gaussian σ to 2. In figure (5) it is reported the probability P to find the particle inside the well after a certain period of time. As expected, the probability decreases as the energy of the wavepacket increases; for energies near the double of the well depth, it stops decreasing and does not reaches 0 beacause part of the transmitted wave escaping at $x = L$ reappears at $x = 0$ entering in the well.

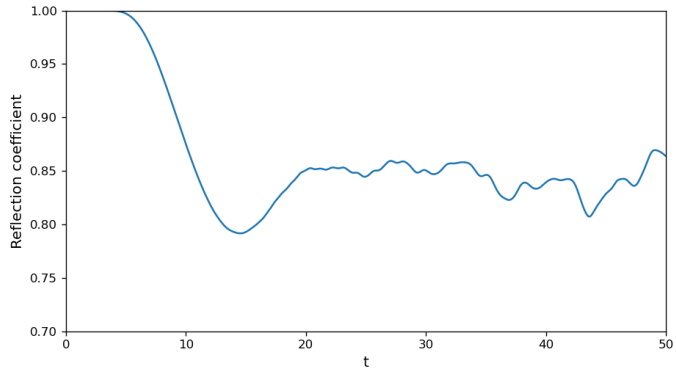

Fig. 4: The probability to find the wavepacket inside the well is less than 1 as it flattens due to the uncertainty principle. The narrower the initial wave, the coarser the precision about the momentum value. This computation results from the situation reported in figure (3); the wavepacket, with a width $\sigma = 0.5$ has a null initial momentum.

The study of the transmission and reflection coefficients is performed also when considering a single potential barrier. In the same way as the potential well, it is considered a wavepacket with increasing energy scattering from the single potential barrier. The results are, as expected, similar to the case of the potential well: the transmission coefficient increases smoothly from 0 to 1 as the energy of the wavepacket increases.

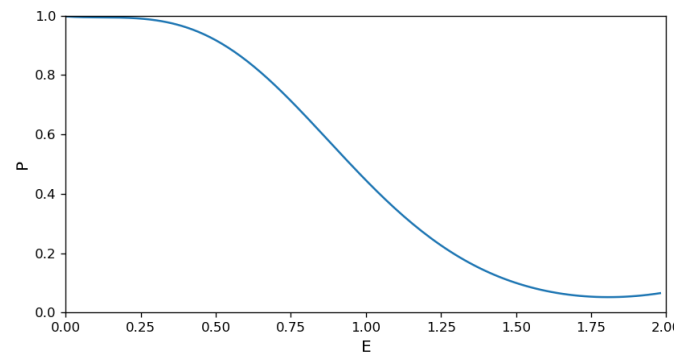

Fig. 5: The probability P to find the wavepacket inside the well as a function of the energy of the particle. At each run, the norm is computed after the same period of time. This is the reason why the probability does not reach 0, as a faster wavepacket (more energy) manages to escape at the endpoint of the interval and enter the well at $x = 0$.

The time spent by the wavepacket inside the barrier is then studied. It is calculated as the difference between the time instant t_0 in which the wavepacket enters the barrier and the time instant t_1 in which it exits: $\tau = t_1 - t_0$. Since the energy is less than the potential, the particle experiences the **quantum tunnelling effect**. In figure (6) τ is plotted as a function of the barrier width l_1. The plot shows for the first values a linear dependence as the wavepacket keeps a constant velocity inside the barrier; then the value of τ departs from this trend as it starts oscillating. The graph still shows an increasing behaviour, but less steep, meaning that somehow the velocity inside the potential region depends on the dimension of the barrier. Studying the transmission coefficient as a function of the barrier width l_1 does not enlighten the situation as T behaves as expected: starting from a unitary value, corresponding to the absence of the

barrier, it quickly decreases until it approaches asymptotically the value 0. The discussion is also complicated by the fact that the transmitted wave is very small, with an intensity decreasing with the width of the barrier: 2.2% for $l_1 = 20$ up to 1.3% for $l_1 = 40$.

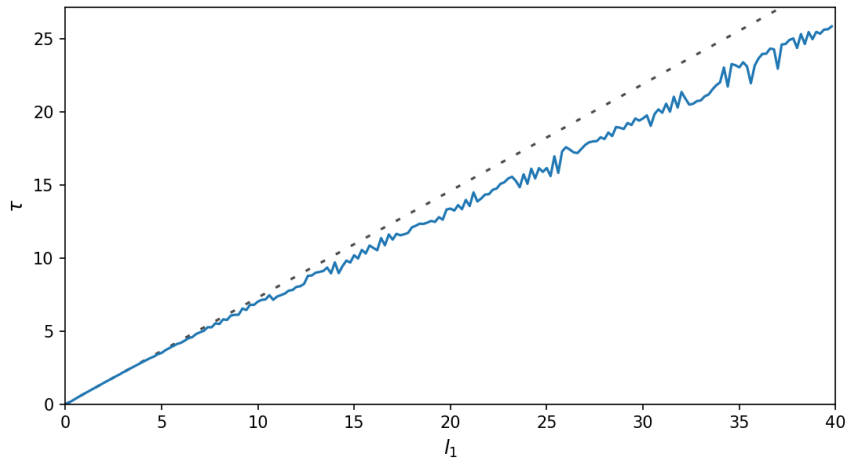

Fig. 6: The time spent tunnelling the barrier is showed as a function of the barrier width l_1. The general behaviour is not a straight line, meaning that the velocity inside the potential region is not constant but depends on the width of the barrier.

Placing a second potential barrier after the first results in similar conclusion as the single barrier case. Here the situation is complicated by the fact that the space interval is periodic and the wave reflected by the first barrier interferes inside the second barrier with the transmitted wave. Therefore, it is considered a larger interval $[0, 180]$; in order to have the same discrete spatial density, N is set to 20000. Since a reflected and transmitted wave originate each time

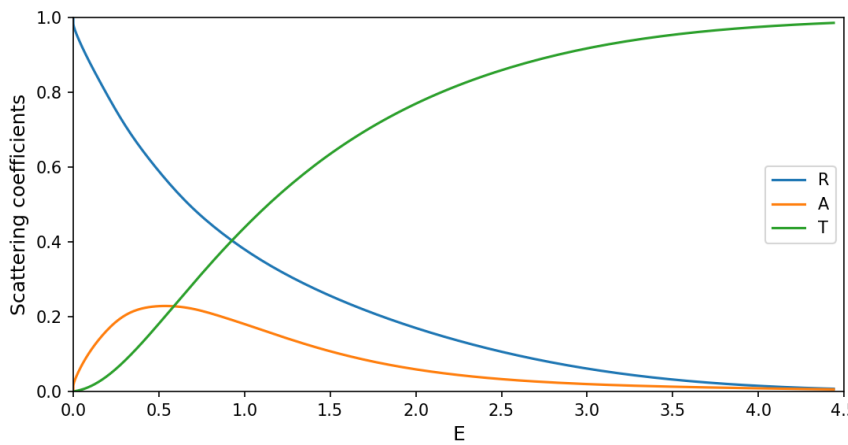

Fig. 7: The computed coefficients for the scattering of the wavepacket with a double potential barrier: the blue line labelled R is the reflection coefficient, while the yellow line (A) corresponds to the absorption coefficient and the green line (T) to the transmission coefficient. Each barrier is tall 1 and large 1; the momentum of the wavepacket is increased by 0.02 after each calculation.

an incident wave encounters a wall of a barrier, it has no meaning to consider one reflected

and one transmitted wave. Thus the coefficient does not stand for the amplitude of the relative waves, but they reflects the probability to find the wavepacket before, inside or after the barrier; figure (7) reports the computed coefficients. As expected, the transmission coefficient smoothly increases with the energy of the wavepacket until it reaches the value 1, as the incident wave is entirely transmitted by the two barriers. The coefficients are computed as soon as the wave entirely exits the second barrier, and thus correspond to different time instants. The absorbtion coefficient represents the part of the wavepacket that is still between the two barriers at the time the computation ended.

5. Conclusion

Since the spatial interval considered is finite and the periodic boundary conditions impose the congruency between the initial and final point, all the results that has been reported depends on the time of the evolution considered: when studying the single barrier scattering, there is a large time interval over which the coefficients remain constants, as showed in figure (2), making easy to select a specific time interval to compute the coefficients; on the other hand, the scattering from the double barrier potential is complicated by the fact that a stationary condition does not exist, for the input parameter considered, and the coefficients are computed at different times corresponding to the transmitted wave entirely exiting the second barrier. The scattering experiments have shown that incident waves encountering a wall of the barrier generates a reflected and a transmitted wave moving in opposite direction; if the barrier is strict enough the waves that originate from the scattering interferes and produce a single reflected and transmitted wave, and not two separate waves one after the other. A wavepacket moving with energy smaller than the potential height get through the barrier by quantum tunnelling; since the time spent from one wall to the other does not show a linear dependence with the width of the barrier, the wavepacket crosses the potential region with a velocity that depends on the dimension of the barrier, in general the speed growing as larger barrier are considered. The curve shown in figure (6) is ideally consistent with an asymptotic behaviour, as if the time spent into the potential region tends to saturate for barrier thick enough, with the wavepacket getting faster and faster (this is known as the **Hartman effect**). The uncertainty principle plays a crucial role in this calculations since the wavepacket spreading contributes to the computed value of the time passed inside the barrier; this could be a factor contributing to the increasing velocity when tunnelling through the barrier. By increasing the energy of the particle, it is greater the fraction of the incident waves reaching the other side; the transmission coefficient grows smoothly with the energy of the particle, not showing any resonance behaviour in correspondence of energy level of the barrier. This is shown in figure (7) and the scattering coefficients behave in the same way also when considering single and double barriers with different dimensions.

The uncertainty principle is involved also when it is considered a wavepacket inside the potential well with zero initial momentum; it is found that there exists a non zero probability for the particle to pass through the barrier and get back in the well by exiting the other wall of the barrier since the space is a monodimensional thorus. This periodic system is a simple modelization of the potential energy felt by an electron moving in a metallic crystal lattice, where there exists a periodic potential generated by the ions. T

The algorithm used to perform the computation is very efficient as the probability is conserved throughout the calculus with a high level of precision (10^{-10}) and almost independent by the spatial density considered or the timestep width. These two parameters rule the accuracy of the simulation and thus the precision of the computed values: a better spatial resolution ensures

that the scattered waves originate and interfere properly, while smaller timesteps prevent the wavepacket from spreading along all the space after an urt with a barrier.

Appendix

A. *Visualize the time evolution – animation.py*

The main program (3.3) eventually writes the computed time evolution into a text file. It is possible to visualize the motion of the wavepacket through a simple Python script, shown in listing (15). The code first reads the evolution file and then it reads the potential file containing two columns with the space interval and the potential arrays. The animation is performed by the matplotlib function `FuncAnimation`.

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation

wave_func = np.loadtxt('Output/moving.txt')   # read the evolution file
num_frames = wave_func.shape[1]   # number of saved waves

V_f = np.loadtxt('Output/pot.txt')   # read potential file
x = V_f[:,0]   # spatial interval
L = np.max(x)
V = V_f[:,1]   # potential
V0 = np.max(V)
V[:] = V[:] / V0 * 1.1 * np.max(wave_func[:,0])   # graphical purposes

fig = plt.figure()
ax = plt.axes(xlim=(0,L), ylim=(0.,np.max(V)*1.1))

ax.set_xlabel('x', fontsize='large')
ax.set_ylabel(r'$|\psi|^2$', fontsize='large')
secax = ax.secondary_yaxis('right')
secax.set_ylabel('Potential', fontsize='large')   # potential energy axis
secax.set_yticks(np.linspace(0., np.max(V), 4))
secax.set_yticklabels(np.around(np.linspace(0., V0, 4), decimals=2))

def ani(t):   # animation function
    ln.set_data(x, wave_func[:,t])
    return ln,

ax.plot(x, V)
ln, = ax.plot([], [])
animat = FuncAnimation(fig, ani, frames=num_frames, interval=20, blit=True)
plt.show()
```

Listing 15: The Python script used to visulaize the motion of the wavepacket after a computation.