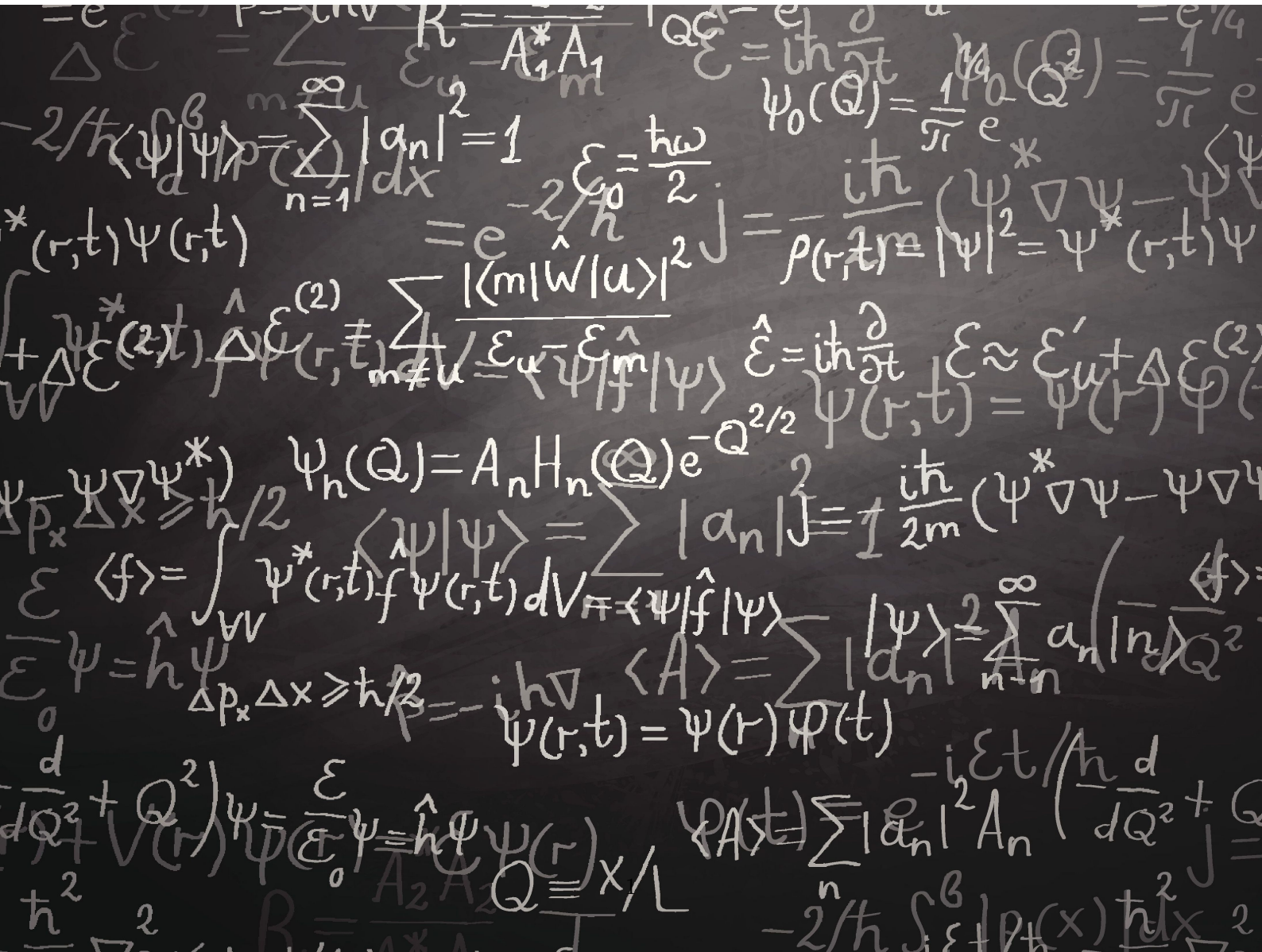
Quantum Physics Lab

Simulation of the Collision of an Electron with a Finite Quantum Square Barrier

Bernat Frangi Mahiques (100454602)

Engineering Physics



Contents

1. Objectives

The aim of this lab is to simulate the temporal evolution of an electron whose wave function is described by a Gaussian wave packet as it collides with a finite square barrier of height V_0 and to analyze the resulting interaction by integrating the 1D time-dependent Schrödinger equation numerically.

2. Introduction

2.1 Scrödinger's Equation: Statement of the Problem

The evolution of the Gaussian packet is governed by Schrödinger's equation:

$$i\hbar\frac{\partial\psi(x,t)}{\partial t} = -\frac{\hbar}{2m_e}\frac{\partial^2\psi(x,t)}{\partial x^2} + V(x)\psi(x,t), \tag{2.1}$$

where m_e is the mass of the electrons. The initial condition is given by:

$$\psi(x,0) = \left(\frac{1}{\pi\sigma_0^2}\right)^{1/4} e^{ik_0x} e^{-\frac{(x-x_0)^2}{2\sigma_0^2}}, \tag{2.2}$$

and the potential function $V(x)$ is:

$$V(x) = \begin{cases} V_0, & \text{if } x \in (0, L) \\ 0, & \text{otherwise} \end{cases} \tag{2.3}$$

where V_0 is the height of the square well and L is its width.

The function $\psi(x,0)$ corresponds to a Gaussian packet that, at time $t = 0$ is centered at x_0, has a spatial width of the order of σ_0, an initial expected velocity $v_0 = \hbar k_0/m_e > 0$ (to the right) and an expected energy of $E_0 = (\hbar^2/2m_e)(k_0^2 + 1/(2\sigma_0^2))$.

2.2 Normalization into Atomic Units

Computers cannot work comfortably with such extreme quantities (very small or very big) as we have in this kind of problem. Therefore, prior to making any calculations on the computer, it is necessary to *move to a more adequate set of units*. For Schrödinger's equation, we typically use **atomic units**:

The transformations that define atomic units are:

$$x \to \overline{x} \equiv \frac{x}{a_0}, \qquad E \to \overline{E} \equiv \frac{E}{E_h}, \qquad t \to \overline{t} \equiv \frac{t}{\hbar/E_h}, \tag{2.4}$$

with $a_0 = 4\pi\epsilon_0\hbar^2/(m_e e^2)$ (Bohr's Radius) and $E_h = \hbar^2/(m_e a_0^2)$.

Once we apply these transformations, the result is that we can express our problem in terms of multiples of those quantities, which are much less extreme and easier for computers to work with. The **normalized Schrödinger equation** becomes:

$$i\frac{\partial\psi(\overline{x},\overline{t})}{\partial\overline{t}} = -\frac{1}{2\overline{m}_e}\frac{\partial^2\psi(\overline{x},\overline{t})}{\partial\overline{x}^2} + \overline{V}(\overline{x})\psi(\overline{x},\overline{t}),$$

with $\overline{V} = V/E_h$ and $\overline{m} = m/m_e$. Notice that $\overline{m}_e = 1$, so:

$$i\frac{\partial\psi(\overline{x},\overline{t})}{\partial\overline{t}} = -\frac{1}{2}\frac{\partial^2\psi(\overline{x},\overline{t})}{\partial\overline{x}^2} + \overline{V}(\overline{x})\psi(\overline{x},\overline{t}), \tag{2.5}$$

2.3 Discretization of the Problem

As we are solving a differential equation numerically in a computer, we must consider both time and space as discrete variables that form a **mesh**. The mesh contains the spatial dimension in the rows, and the temporal dimension along the columns. At each point (j,k) of the mesh, the value of some magnitude is given for the position corresponding to the row j at the time corresponding to the column k.

- **Space:** space is a discrete variable that will take the values

  $$x_{min},\ x_{min} + \Delta x,\ x_{min} + 2\Delta x,\ ...,\ x_{min} + (M-1)\Delta x,$$

  where $x_{min} + (M-1)\Delta x = x_{max}$. Therefore, there will be M spatial points x_j $(j = 0, 1, ..., M-1)$ and the mesh will have M rows.

- **Time:** time is a discrete variable that will take the values

  $$0,\ \Delta t,\ 2\Delta t,\ ...,\ (N-1)\Delta t.$$

  Therefore, there will be N temporal points t_k $(k = 0, 1, ..., N-1)$ and the mesh will have N columns.

Once we have defined the geometry of the space-time mesh, we can easily store the values of the discretized wave function in that mesh, where we have a $j \times k$ matrix with elements $\psi_{jk} \equiv \psi_j^k \equiv \psi(x_j, t_k)$ for $j = 0, 1, ..., M-1$ and $k = 0, 1, ..., N-1$.

2.4 Discretization of the Derivatives

We use the approach known as finite differences in order to calculate the two derivatives that appear in the Schrödinger equation.

- **Temporal Derivative:** We can approximate the time derivative of the wave function at each point (j, k) of the space-time mesh using the expression:

$$\frac{\partial \psi(x_j, t_k)}{\partial t} \equiv \dot{\psi}_j^k = \frac{\psi_j^{k+1} - \psi_j^k}{\Delta t} \tag{2.6}$$

- **Spatial Second Derivative:** We can approximate the spatial second derivative of the wave function at each point (j, k) of the space-time mesh using the expression:

$$\frac{\partial^2 \psi(x_j, t_k)}{\partial x^2} \equiv \psi_j''^k = \frac{\psi_{j+1}^k - 2\psi_j^k + \psi_{j-1}^k}{(\Delta x)^2} \tag{2.7}$$

2.5 Integration Method

One method that is often used for solving this problem is the **Crank-Nicholson method**. It gives us a way of obtaining the wave function for the next temporal step by using the previous. The expression that is used in this method is:

$$\dot{\psi}_j^k = \frac{i}{4}\left(\psi_j''^{k+1} + \psi_j''^k\right) - \frac{i}{2} V_j \left(\psi_j^{k+1} + \psi_j^k\right), \tag{2.8}$$

where V_j is the potential function evaluated at position x_j. If we expand this so that all the dependencies on the future are on the same side, and the dependencies on the present are on the other side, we obtain:

$$\begin{aligned} \psi_j^{k+1} - \frac{i\Delta t}{4(\Delta x)^2}\left(\psi_{j+1}^{k+1} - 2\psi_j^{k+1} + \psi_{j-1}^{k+1}\right) + \frac{i}{2}\Delta t V_j \psi_j^{k+1} = \\ = \psi_j^k + \frac{i\Delta t}{4(\Delta x)^2}\left(\psi_{j+1}^k - 2\psi_j^k + \psi_{j-1}^k\right) - \frac{i}{2}\Delta t V_j \psi_j^k \end{aligned} \tag{2.9}$$

If we write this in vector form, we obtain the equation:

$$\mathbf{L} \cdot \Psi^{k+1} = \mathbf{R} \cdot \Psi^k \tag{2.10}$$

where:

$$\mathbf{L} = \begin{pmatrix} \beta + i\Delta t V_1/2 & -i\alpha & 0 & \dots & 0 & 0 \\ -i\alpha & \beta + i\Delta t V_2/2 & -i\alpha & \dots & 0 & 0 \\ 0 & -i\alpha & \beta + i\Delta t V_3/2 & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \dots & \beta + i\Delta t V_{M-1}/2 & -i\alpha \\ 0 & 0 & 0 & \dots & -i\alpha & \beta + i\Delta t V_M/2 \end{pmatrix}$$

$$\mathbf{R} = \begin{pmatrix} \beta^* - i\Delta t V_1/2 & i\alpha & 0 & \dots & 0 & 0 \\ i\alpha & \beta^* - i\Delta t V_2/2 & i\alpha & \dots & 0 & 0 \\ 0 & i\alpha & \beta^* - i\Delta t V_3/2 & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \dots & \beta^* - i\Delta t V_{M-1}/2 & i\alpha \\ 0 & 0 & 0 & \dots & i\alpha & \beta^* - i\Delta t V_M/2 \end{pmatrix}$$

and where $\alpha = \Delta t/4(\Delta x)^2$ and $\beta = 1 + 2i\alpha$.

2.6 Solving Tridiagonal Matrix Problems

We can write the system in **Equation 2.10** as:

$$\mathbf{A}x = s, \tag{2.11}$$

where $\mathbf{A} = \mathbf{L}$, $x = \psi^{k+1}$ and $s = \mathbf{R} \cdot \psi^k$. Since the matrix $\mathbf{A}$ is tridiagonal, there is a simple recursive method that can be used to find the solution.

First, we need to find the *recursive coefficients*, with:

$$a'_i = \begin{cases} \frac{a}{d_1}, \ i = 1 \\ \frac{a}{d_i - a \cdot a'_{i-1}}, \ i = 2, 3, ..., M-1 \end{cases} \tag{2.12}$$

and:

$$s'_i = \begin{cases} \frac{s_1}{d_1}, \ i = 1 \\ \frac{s_i - a \cdot s'_{i-1}}{d_i - a \cdot a'_{i-1}}, \ i = 2, 3, ..., M-1, M \end{cases} \tag{2.13}$$

Finally, to find the vector x, we need to apply the following recursive pattern:

$$x_i = \begin{cases} s'_M, \ i = M \\ s'_i - a'_i \cdot x'_{i+1}, \ i = M-1, M-2, ..., 2, 1 \end{cases} \tag{2.14}$$

3. Results and Analysis

The code for this lab can be found in this is GitHub repository. All tasks have been solved in just one file called `simulation.py`. Another file named `initial_parameters.py` has been used to store all the initial parameters of the main simulation done in **Section 3.1**. A last file has been used to define some of the functions used in the main program and then imported as a custom library, in order to have a less cluttered main file. This last file is named `quantumpython.py`.

Note: in order to use the python programs, the following python libraries should be manually installed: `matplotlib`, `numpy`, `rich` *and* `progressbar`. *Also, this project has been made using* `python 3.8.5`.

3.1 Animation of the Interaction

In this section, the animation of the interaction of the wave function of an electron with a square barrier has been made. It is available in this link as an `.mp4`, and the wave function for some of the times is also plotted in **Figure 1**. The initial parameters used for this simulation are shown in **Table 1** and also stored in `initial_parameters.py`.

Parameter	value
Δt	0.005
t_{final}	50
Δx	0.05
x_{min}	-60
x_{max}	60
k_0	1
σ_0	3
x_0	-10
L	2
V_0	2
M	2401

Table 1: Parameters values to be used for the first simulation.

In order to use the program `simulation.py` to generate the movie in the link and the graph in **Figure 1**, the following steps should be followed:

1. Execute the program with `python3 simulation.py` while in the directory where the file `simulation.py` is stored.
2. You will be prompted to choose the number of time points that should be computed and also the time step that should be used. Enter those values (recommended: $N =$

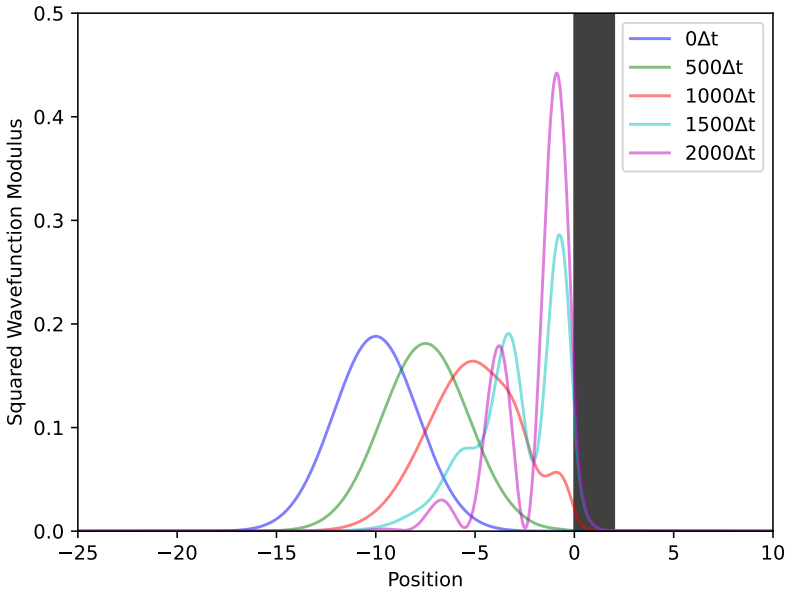


Figure 1: Wave Function at $0\Delta t$, $500\Delta t$, $1000\Delta t$, $1500\Delta t$ and $2000\Delta t$.

5000, $\Delta t = 0.005$, in normalized units). Default values can be selected by pressing enter.

3. a) If this is the **first time** that you have executed this program, you will need to compute the wave function from scratch. To do this, you need to choose option 1 in the first menu that will appear (Compute the wave function from scratch using the initial parameters from file `initial_parameters.py`).

   The program will then compute the matrix of the wave function and the matrix of the modulus squared of the wave function. It will then export both matrices to separate text files so that they can be imported in future executions of the program to avoid having to repeat calculations each time.

3. b) If you have the **matrices already exported to text files**, there is no need to compute them again. In that case, you can select the second option in the first menu (Import the wave function from the file `wavefunction.txt`). This will make things much quicker, especially for large N. Just make sure that the values of N and Δt that have been chosen at the start of the program are the same that have been used to create the exported files, otherwise you may encounter errors!

4. Once the program has calculated or imported the wave function, another menu will appear. In order to create the animation of the evolution of the wave function and its interaction with the square barrier, select option 1 (Create animation of the evolution of the wave function in time). When the program asks if you would like to plot the transmission probability, choose `no`. To plot the graph of **Figure 1** instead, choose option 2 (Plot the wave function for $0\Delta t$, $500\Delta t$, $1000\Delta t$, $1500\Delta t$ and $2000\Delta t$).

5. Whether you have chosen option 1 or 2 in the second menu, you will be given the option to save the animation/figure to your disk. You can choose to do so or not, but in the case you choose to export, you must make sure that you have a folder named

`Output-Media` in the same directory where the `simulation.py` file is located, or you *will* get an error. This is the folder where the output files will be stored.

3.2 Transmission Probability

In this section, the probability of transmission of the wave function has been calculated as a function of time. The resulting graph is shown in **Figure 2**.

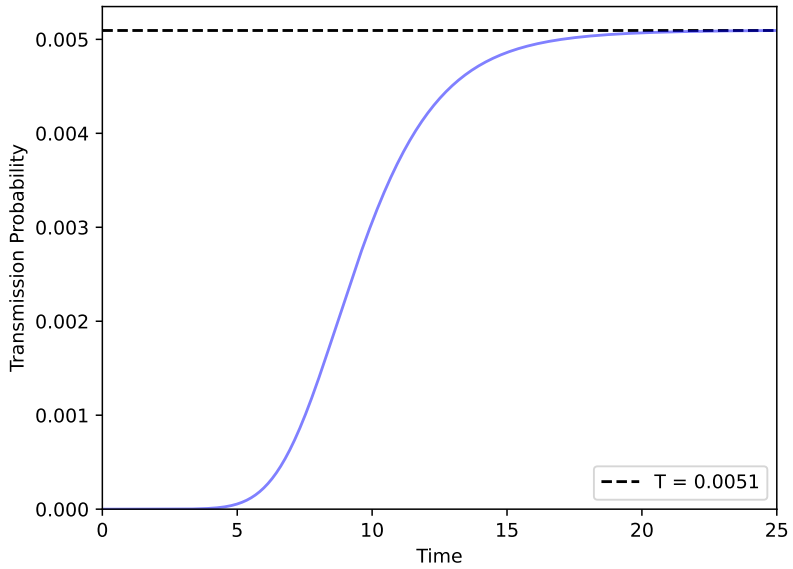


Figure 2: Probability of Transmission of the Wave Function as a Function of Time

In this case, we can see that the asymptotic value of the transmission probability is:

$$\boxed{T_{asymptotic} = 0.0051} \tag{3.1}$$

In order to use the program `simulation.py` to generate the graph in **Figure 2**, the following steps should be followed:

1. Follow steps 1 to 3 of **Section 3.1**.

4. Once the program has calculated or imported the wave function, another menu will appear. Choose option 4 (Compute probability of finding the electron beyond the barrier as a function of time). The plot of **Figure 2** will be shown and also saved in the `Output-Media` directory.

3.3 Dependence of the Asymptotic Transmission Probability on k_0

For this study, several simulations have been carried out for different values of k_0 varying from 0.5 to 8 (with all other parameters being the same), and their transmission probabilities have been plotted versus time. Then, the asymptotic transmission probability has then been calculated for each case and plotted versus the energy $E_0 = (k_0^2 + 1/(2\sigma_0^2))/2$.

The two plots obtained are shown in **Figure 3** and **Figure 4**.

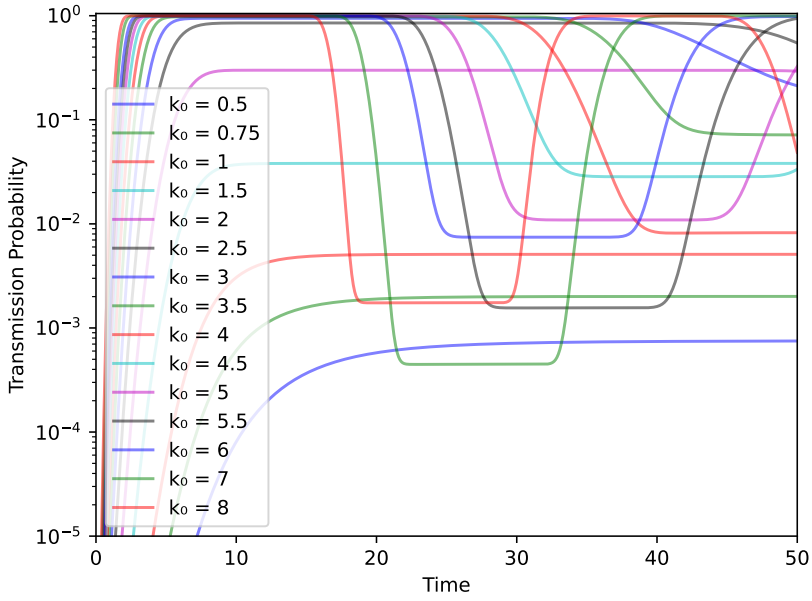


Figure 3: Transmission Probability vs. Time for Different Values of k_0.

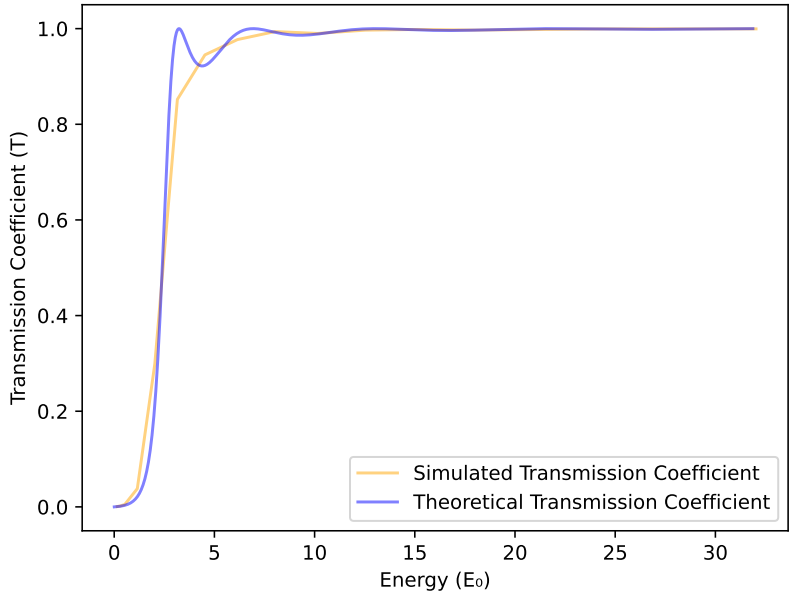


Figure 4: Transmission Probability as a Function of E_0

In order to use the program `simulation.py` to generate the graphs in **Figure 3** and

Figure 4, the following steps should be followed:

1. Execute the program with `python3 simulation.py` while in the directory where the file `simulation.py` is stored.

2. Again, you will be prompted to choose N and Δt. In this case, the recommended initial parameters for time are $N = 10000$ and $\Delta t = 0.005$, so that the whole trend of T is shown for all the values of k_0. Default values can be selected by pressing enter.

4. In this program we will not use the wave function computed in the previous sections, so we can skip the computation/import of the wave function, as we will calculate the appropriate wave functions later. Therefore, once the first menu appears, choose option 4 (Skip).

5. When the next menu appears, choose option 5 (Compute probability of finding the electron beyond the barrier for different values of k_0).

6. The program will now ask for the values of k_0 for which you would like to make the computations. Enter those values separated by commas (e.g. `1, 1.5, 2, 3`). You can also press enter to choose the default values, which are `0.5, 0.75, 1, 1.5, 2, 2.5, 3, 3.5, 4, 4.5, 5, 5.5, 6, 7, 8`. For this plot, choose the default values.

7. The program will then proceed to calculate the wave function, the modulus squared of the wave function and the probability of transmission for each of the k_0. This may take some time, but in the end a plot of the data like the one in **Figure 3** will appear and will also be saved to the `Output-Media` directory.

8. You will then be given the option to study the dependence of the (asymptotic) transmission probability on the energy E_0. If you want to generate the plot in **Figure 4**, choose `Y` to accept. The program will then make the plot and store it in the `Output-Media` directory.

There are a few interesting observations that we can make by looking at the plot of **Figure 3**. We can see that the curves corresponding to the smallest values of k_0 are increasing up to the asymptotic transmission probability and they stay near that value forever. On the other hand, we can also observe that the larger k_0 curves have a different behaviour: they increase to their first asymptotic value, then decrease and increase again. This pattern would be continued on and on.

In order to understand this, we must remember that the simulation that we have made has the imposed boundary conditions that state that the wave function is zero at both extremes of the spatial domain that we have considered. This means that the wave function is "reflected" off the sides of the domain whenever it collides with them. Then, it starts moving in the opposite direction. In that sense, the boundaries of the space domain behave like some kind of infinite square barrier. The result is that the wave function travels *several times* through the finite square barrier that we are analysing.

With this in mind, lets consider the case of **small $\mathbf{k}_0$**. In this link, we can see a plot which shows both the evolution of the wave function (with $k_0 = 1$) and its transmission probability plotted at the same time, so that we can see the relation between the two. This plot is generated by following the steps in **Section 3.1** but choosing *option 1* and `yes` in step 4.

As we can see in the animated plot, the wave function reaches the square barrier and is then mostly reflected, although part of the wave function goes through. This is because the packet has $E < V_0$, so the part of the wave that goes through the barrier does it through tunneling. After the interaction happens, both the transmitted part of the wave and the reflected part move away from each other and towards the spatial domain boundaries.

If we carried on the simulation, we would see both parts "rebound" on those boundaries and go back towards the square barrier. Tunneling would happen again and the pattern would be repeated on and on. Overall, the trend would see the transmission probability increase (slightly irregularly) as tunneling happens between both sides. There would eventually come a point where the trend would reverse, and the transmission probability would slowly and irregularly start to decrease. This overall pattern would be repeated indefinitely, until some sort of equilibrium is reached. **Figure 5** shows a plot of the transmission probability as a function of time generated by the program `simulation.py` for $N = 5000$, $\Delta t = 0.5$ and $k_0 = 1$.

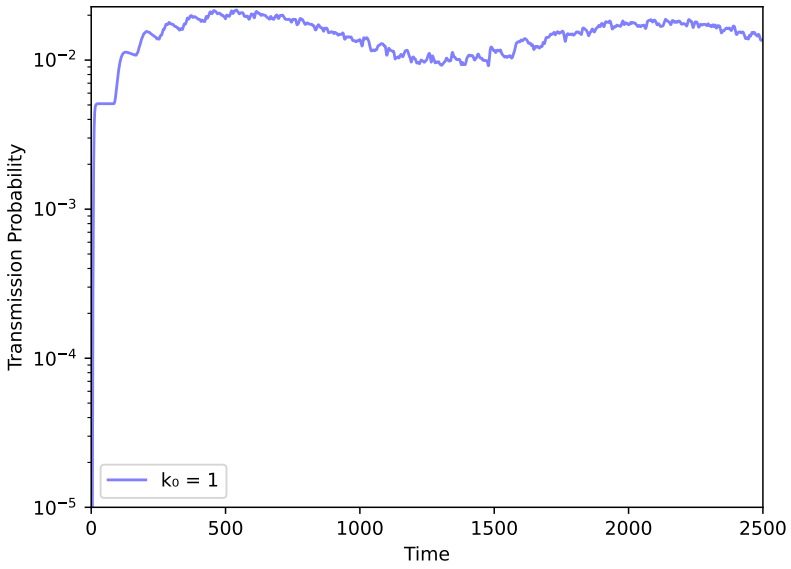


Figure 5: Transmission Probability as a Function of t for $N = 5000$, $\Delta t = 0.5$ and $k_0 = 1$

Now, lets consider the case of **large $\mathbf{k}_0$**. In this link, we can see a similar plot to the previous one but for $k_0 = 8$. As we can see in the animated plot, the wave function reaches the square barrier and, contrary to what happened before, it goes through almost completely (this is because it has a lot more energy). Therefore, the transmission probability increases to almost one. Eventually, the wave function reaches the end of the spatial domain and changes direction, crossing the square barrier again and causing the transmission probability to decrease. As before, a portion of the wave function stays behind and does not go through

the square barrier, so the transmission probability does not decrease all the way to zero. If the wave function were to reflect on the spatial domain boundary again and traverse the square barrier for a third time, the transmission probability would again increase. On return of the wave function back through the square barrier for a fourth time, it would decrease again to a value a little bit higher than the previous decrease.

Figure 6 shows a plot of the transmission probability as a function of time for generated by the program `simulation.py` for $N = 5000$, $\Delta t = 0.5$ and $k_0 = 5$.

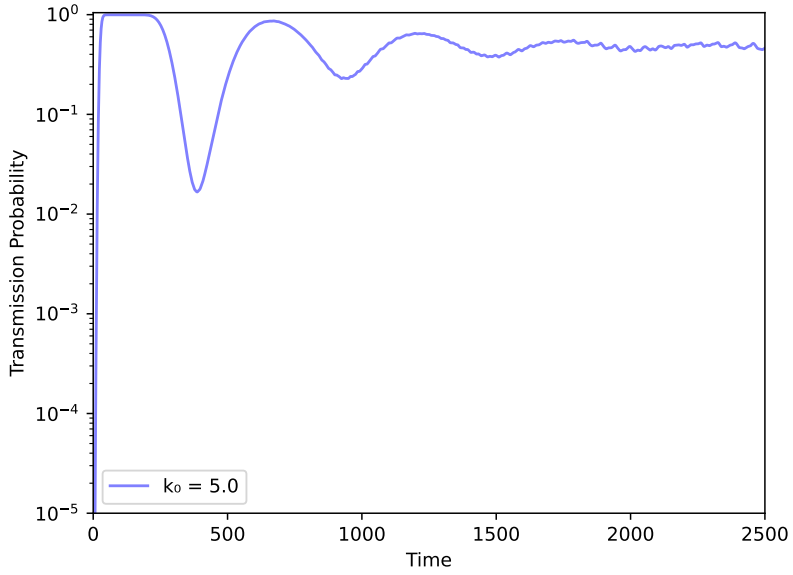


Figure 6: Transmission Probability as a Function of t for $N = 5000$, $\Delta t = 0.5$ and $k_0 = 5$

Of course, in a real case we would consider an infinite spatial domain, so all these phenomena would not occur. Rather, the reflected and transmitted portions of the wave function would continue unhindered on their way forever, and never again touch the square barrier. Therefore, in a real case the asymptotic transmission probabilities would be maintained forever.

Moving on to **Figure 4**, We can see that the simulated transmission coefficient follows the general trend of the theoretical one for planar waves. There is, however, one notable difference: the oscillations are not so clear in the simulated plot, especially the first maximum where resonant transmission takes place.

The obvious reason for this difference is that our simulation is done for a Gaussian wave packet, and not for a single plane wave. If we wanted to make the oscillations more clear in the plot from our simulation, we could increase the value of σ_0 to make the initial condition of our Gaussian wave packet more similar to the initial condition of a plane wave:

$$\psi(x,0) = \left(\frac{1}{\pi\sigma_0^2}\right)^{1/4} e^{ik_0x} e^{-\frac{(x-x_0)^2}{2\sigma_0^2}} \xrightarrow{\text{for large } \sigma_0} Ae^{ik_0x}, \tag{3.2}$$

where:

$$A = \left(\frac{1}{\pi\sigma_0^2}\right)^{1/4} \quad \text{and} \quad e^{-\frac{(x-x_0)^2}{2\sigma_0^2}} \sim 1$$

So, for large σ_0, the initial condition is approximately $\psi(x,0) \sim Ae^{ik_0x}$ (especially for small x), which is an initial condition corresponding to a plane wave. If we run the program `simulation.py` following the steps in **Section 3.3** but with `sigma_0 = 7` in the file `initial_parameters.py` instead of `sigma_0 = 3`, we obtain the plot in **Figure 7** (we also need to change `x_0 = -10` to `x_0 = -20` because the Gaussian wave packet is now wider, so it should start further back). In this plot, we have included the following values of k_0: 0.5, 0.75, 1, 1.25, 1.5, 1.75, 2, 2.1, 2.25, 2.5, 2.75, 3, 3.5, 4, 4.5, 5, 5.5, 6, 7, 8.

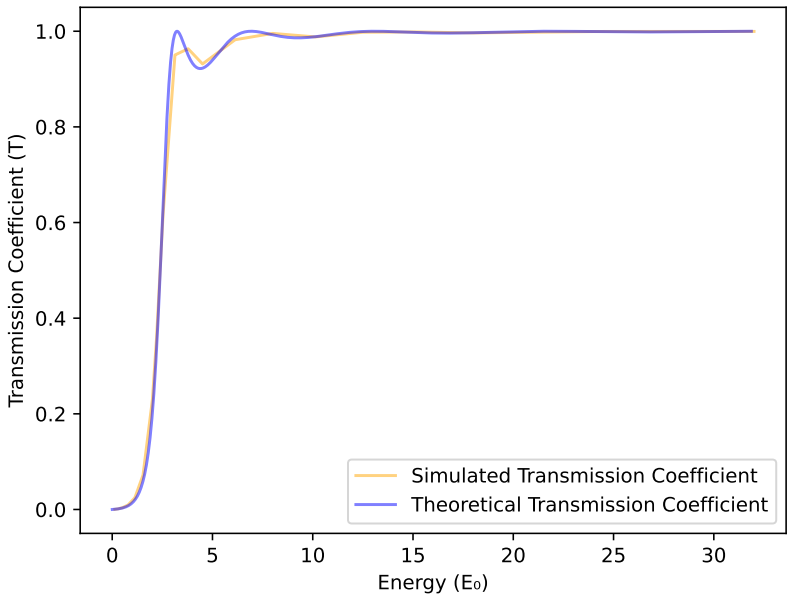


Figure 7: Transmission Coefficient as a Function of E_0 for a Gaussian Wave Packet With $\sigma_0 = 7$.

Clearly, this plot is much closer to the transmission probability corresponding to the plane waves. Another thing we could do to improve the closeness between both plots is increase the number of k_0 values so that there are more data points and the resolution is better.

3.4 Analysis of Accuracy

In order to analyse the accuracy of the wave function calculated using this python code, the area under the probability distribution function has been integrated. The theoretical value of this integral is one, as the total probability must be one:

$$\int_{-\infty}^{\infty} |\psi(x,t)|^2 dx = 1, \ \forall t \tag{3.3}$$

In order to compute this integral with the code `simulation.py`, follow the steps 1 to 3 in **Section 3.1**. A file called `integral_of_the_modulus_squared.txt` will be created with the value of the integral for all the times. In the second menu, choose option 3 (Analyse the accuracy of the method). This will compute the average value of the integral and also the standard deviation with respect to the theoretical value, 1.

In the case with the initial parameters of **Section 3.1**, we have obtained the following values for the average and standard deviation of the integral in **Equation 3.3** (output is also shown in **Figure 8**):

$$\text{Average of } \int_{-\infty}^{\infty} |\psi(x,t)|^2 dx = 0.9999999997131068 \tag{3.4}$$

$$\text{Standard Deviation of } \int_{-\infty}^{\infty} |\psi(x,t)|^2 dx \text{ (with respect to 1)} = 3.34733659674436 \cdot 10^{-10} \tag{3.5}$$

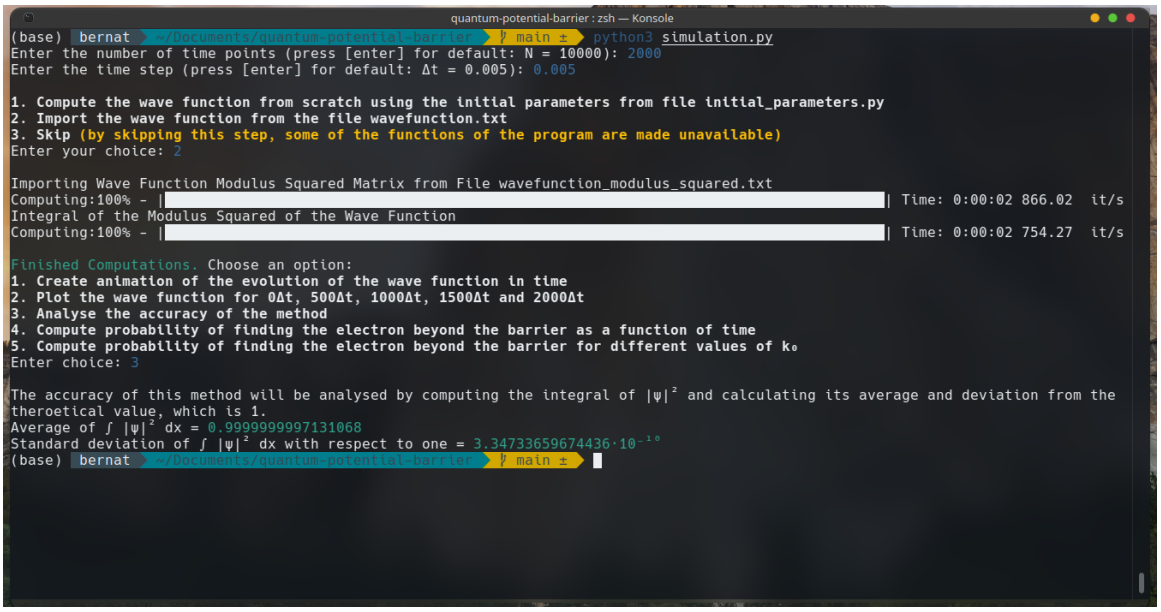


Figure 8: Analysis of the Accuracy of the Results

From this data, we can see that the integral is almost exactly 1 for all times, with a very small standard deviation, of the order of 10^{-10}. This verifies that the simulation is indeed extremely accurate.

3.5 Solving this Same Problem Using Euler's Method

There is another file in the GitHub repository (called `simulation_euler.py`) that was also elaborated in this lab to quickly and graphically show the difference between the results obtained with the *Crank-Nicholson method* and those obtained with *Euler's method*. It simply computes the wave function using the initial parameters from the `initial_parameters.py`

file and plots its modulus squared in an animation that quickly explodes and becomes completely useless for accurately extracting any of the conclusions that we were trying to obtain in this lab. As the wave function is being computed, many warnings appear in the console. This is due to the large values that the wave function quickly obtains as it explodes.

This shows the importance of the little change in the expressions used in both methods. The one used in Euler's Method is shown below, whereas the one used the Crank-Nicholson method is shown in **Equation 2.8**:

$$\dot{\psi}_j^k = \frac{i}{2}\psi''^k_j - iV_j\psi_j^k \tag{3.6}$$

4. Conclusions

After having done this lab, there are several conclusions that we can extract:

Firstly, it is possible to discretize the problem of *Schrödinger's equation* and solve it for any potential using the *Crank-Nicholson method*. Furthermore, this method gives very accurate results as opposed to *Euler's method*, which explodes very quickly, rendering it useless after not many iterations. We have been able to further test and verify the accuracy of the *Crank-Nicholson method* by integrating the area under the probability density function of the wave function for all steps of the simulation and finding it to have a standard deviation only of the order of 10^{-10} with respect to the theoretical value, which was 1.

Secondly, we have been able to create an animation of the interaction of the wave function with a square potential barrier using the Crank-Nicholson method. With this animation, we have been able to compute the transmission probability of the wave function and its asymptotic value. We have seen that, for $k_0 = 1$, the transmission probability increases gradually with time as the wave function approaches the barrier. Then, T goes to the asymptotic value until the end of the simulation.

We have seen that this is common in all animations with low k_0: the transmission probability T increases until the asymptotic value and then stays close to that value forever, with some irregular fluctuations in the longer term. On the other hand, for high k_0, we have seen that T suffers some much more noticeable increases and decreases in the short term, but has a similar behaviour to that of small k_0 in the longer term.

We have deduced that this phenomenon that we observe - the fluctuation of the transmission probability - is only present due to the finiteness of our spatial domain. If the domain was infinite, the interaction of the wave function with the square barrier would happen only once and we would not have these fluctuations.

Thirdly, we have analysed the dependence of the asymptotic transmission probability on the energy E_0 (determined with k_0). We have seen that the transmission probability is not always either 0 or 1, as in the classical case, but increases gradually from one to the other. In addition, there are fluctuations of T even for large E_0, although these become increasingly weaker.

Lastly, we have found that we can make this simulated dependence of T on E_0 more similar to that of a plane wave by increasing σ_0 of the Gaussian wave packet.

This lab has helped to consolidate knowledge and understanding of quantum physics, particularly the phenomenon of square quantum barriers and wave function interactions.