Computational Physics - Exercise 8 TDSE

Christian Gorjaew (354259)
christian.gorjaew@rwth-aachen.de

Julius Meyer-Ohlendorf (355733)
julius.meyer-ohlendorf@rwth-aachen.de

June 30, 2020

1. Introduction

In the following we solve the time-dependent Schrödinger equation (TDSE) in 1D using the `Second-order product formula algorithm`. The case of a particle moving in free space and the case of a particle impinging on a potential barrier is considered. The time evolution of the probabilities is presented and compared.

2. Simulation model and method

Model:
In quantum mechanics the TDSE describes the time evolution of the wave function Φ. In 1D the non-relativistic TDSE for a single particle is given by:

$$i\hbar\frac{\partial}{\partial t}\Phi(x,t) = \underbrace{\Big(-\frac{\hbar^2}{2M}\frac{\partial^2}{\partial x^2} + V(x)\Big)}_{H}\Phi(x,t), \tag{1}$$

with the Hamilton Operator H, where $V(x)$ corresponds to the potential, M to the mass of the particle and $\hbar$ to the reduced Planck constant. In this exercise we set $M = \hbar = 1$. The probability density is calculated as $\rho(x,t) = |\Phi(x,t)|^2$ with the normalization condition $\int \rho(x,t)dx = 1$.
We model two different cases in 1D using a simulation box in x direction of length 100. In the first case we solve the TDSE for a particle moving in free space with kinetic energy $E_{\text{kin}} = 1/2$. The potential is set to $V(x) = 0$ in the whole simulation box.
In the second case, a particle with the same kinetic energy impinging on a rectangular potential barrier with energy height $V = 2$ and width $d = 0.5$ is considered. For that, we set:

$$V(x) = \begin{cases} 0 & \text{if} \quad 0 \le x < 50 \\ 2 & \text{if} \quad 50 \le x \le 50.5 \\ 0 & \text{if} \quad 50.5 < x \le 100 \end{cases} \tag{2}$$

In both cases a Gaussian wave packet centered at position $x_0 = 20$ is used as the initial wave function at time $t = 0$:

$$\Phi(x,t=0) = \Big(\frac{1}{2\pi\sigma^2}\Big)^{\frac{1}{4}} e^{iq(x-x_0)}e^{-(x-x_0)^2/4\sigma^2}, \tag{3}$$

with width $\sigma = 3$ and wave vector $q = 1$.

Method:
The formal solution of the TDSE is given by:

$$\Phi(x,t) = e^{-itH}\Phi(x,t=0), \tag{4}$$

with the time evolution operator e^{-itH}.
To solve the TDSE numerically, the potential, the wave function and the second derivative

in space are discretized on the lattice with spacing Δ. The discretization leads to a matrix equation:

$$i\frac{\partial}{\partial t}\begin{pmatrix}\Phi_1(t)\\ \Phi_2(t)\\ \Phi_3(t)\\ \vdots\\ \vdots\\ \Phi_L(t)\end{pmatrix} = \Delta^{-2}\underbrace{\begin{pmatrix}1+\Delta^2V_1 & -1/2 & 0 & & & 0\\ -1/2 & 1+\Delta^2V_2 & -1/2 & & & \\ 0 & -1/2 & 1+\Delta^2V_3 & & & \\ & & & \ddots & & 0\\ 0 & & & & 1+\Delta^2V_{L-1} & -1/2\\ & & & 0 & -1/2 & 1+\Delta^2V_L\end{pmatrix}}_{\boldsymbol{H}}\begin{pmatrix}\Phi_1(t)\\ \Phi_2(t)\\ \Phi_3(t)\\ \vdots\\ \vdots\\ \Phi_L(t)\end{pmatrix}$$

We use a spacing of $\Delta = 0.1$ and set the number of lattice points to $L = 1001$. Discretizing time with a time step τ (in this exercise τ=0.0001) the wave function at $t + \tau$ is given by:

$$\Phi(t+\tau) = e^{i\tau\boldsymbol{H}}\Phi(t) \tag{5}$$

The matrix exponential can only be calculated if the discretized Hamilton matrix $\boldsymbol{H}$ can be diagonalized. To avoid this problem, the exact time evolution operator $e^{i\tau\boldsymbol{H}}$ is approximated by an approximate time-step operator $\boldsymbol{U}$. At first the exact time evolution operator is decomposed $\boldsymbol{H} = \boldsymbol{A}+\boldsymbol{B}+\boldsymbol{C}$, where $\boldsymbol{A}$, $\boldsymbol{B}$ and $\boldsymbol{C}$ are chosen so that their matrix exponentials can be calculated. Here, $\boldsymbol{H} = \boldsymbol{V}+\boldsymbol{K_1}+\boldsymbol{K_2}$ was chosen (c.f. lecture 18, slide 49). The exact time evolution operator is then approximated as follows:

$$e^{-i\tau\boldsymbol{H}} \approx \boldsymbol{U} = e^{-i\tau\boldsymbol{K_1}/2}e^{-i\tau\boldsymbol{K_2}/2}e^{-i\tau\boldsymbol{V}}e^{-i\tau\boldsymbol{K_2}/2}e^{-i\tau\boldsymbol{K_1}/2}, \tag{6}$$

where each matrix exponential is easily calculated (c.f. lecture 18, slide 50 ff.). Repeatedly applying this approximate time evolution operator to the initial wave function yields the time evolution of the wave function and defines the `Second-order product formula algorithm`. We performed $m = 50000$ applications of the approximate time evolution operator corresponding to a simulation time of $t_{\text{sim}} = m\tau = 50$. In order to preserve the total probability, $\boldsymbol{U}$ is chosen to be unitary which also implies that this algorithm is unconditionally stable.

3. Simulation results

The `python` code used for the simulation can be found in the Appendix A.
The evolution of the probability at different times in the case of no potential barrier is shown in figure 1. Additionally, an animation of the evolution can be found under the link in the footnote.[1]. On the discretized lattice the probability is calculated as $P(x,t) = |\Phi(x,t)|^2\Delta = \rho(x,t)\Delta$. This corresponds to the probability of finding the particle at time t in a space region Δ around x. Starting from a Gaussian wave packet, hence a Gaussian probability function $P(x,t)$

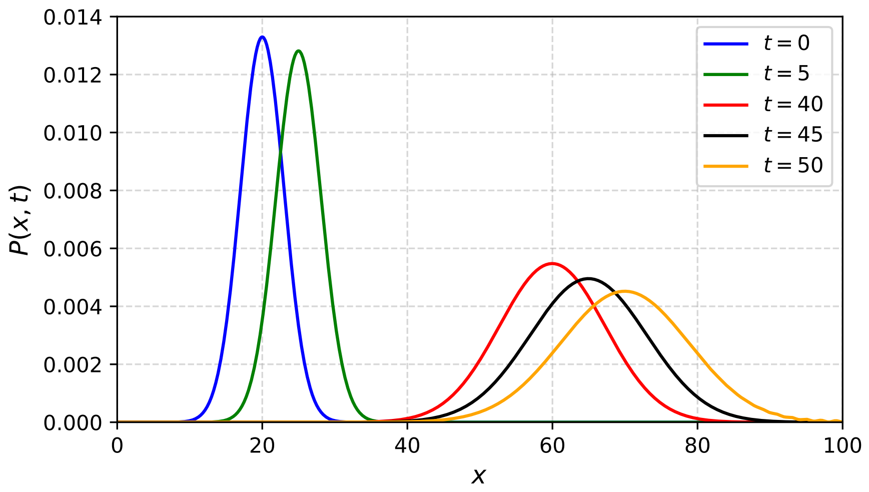

Figure 1: Simulated probability $P(x,t)$ at different times in the case of no potential barrier. Simulation parameters: $\Delta = 0.1$, $L = 1001$, $\tau = 0.0001$, $m = 50000$.

at $t = 0$, centered at $x_0 = 20$ (c.f. equation (3)) the wave packet moves in the positive x-direction. It is clearly visible that the spread in space of the probability continuously increases and reaches its maximum (inside the simulation time interval) at the end of the simulation $t = 50$. This behaviour is expected and can be explained as follows. The initial Gaussian wave packet is constructed by a Fourier Transform corresponding to the superposition of plane waves with different momenta. It can then be shown (c.f. Theoretical Physics 3) that the spread in space is time dependent and increases continuously.
Our results of the second case with a potential barrier of height $V = 2$ and width $d = 0.5$ are shown in figure 2. Again, an animation can be found under the footnote 1. For better visibility the probability for $x > 50.5$ is normalized as soon as the probability at the first lattice point to the right of the barrier surpasses a threshold of $P_{\text{thres}} = 0.0001$. This ensures, that the probability to the right of the potential barrier does not explode in the beginning of the simulation.
In classical physics a particle with E_{kin} can only overcome a potential step with energy V and width d if $E_{\text{kin}} > V$. In a quantum mechanical system the particle can also overcome a potential step if $E_{\text{kin}} < V$. The probability of overcoming the potential step is quantified by the transmission coefficient T. For $E_{\text{kin}} < V$ it is given by [2]:

$$T = \frac{1}{1 + \frac{V^2 \sinh^2(d)}{4E_{\text{kin}}(V - E_{\text{kin}})}} \tag{7}$$

Hence, the transmission coefficient increases for higher kinetic energies and therefore for higher

[1]Animated simulation results: `https://rwth-aachen.sciebo.de/s/jA8FPIZ87r1OiAY`.To see the animation, you might have to download the files.
[2]c.f., `https://en.wikipedia.org/wiki/Rectangular_potential_barrier`

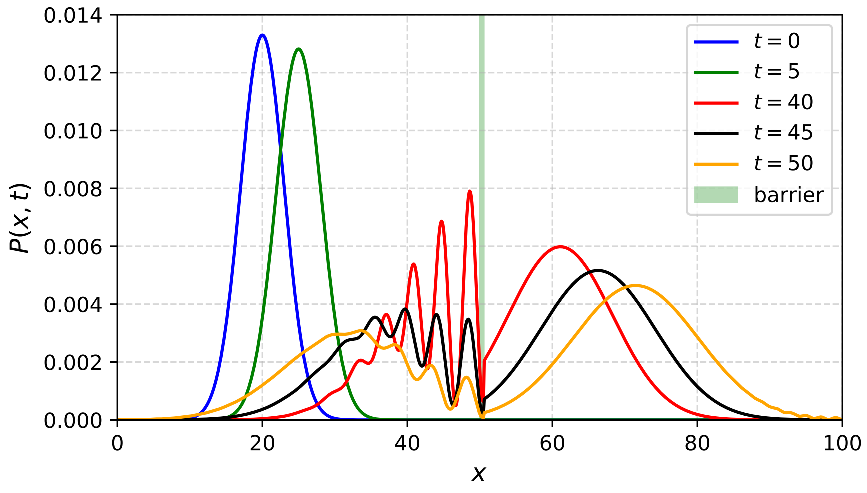

Figure 2: Simulated probability $P(x,t)$ at different times in the case of a potential barrier with energy height $V = 2$ and width $d = 0.5$. Simulation parameters: $\Delta = 0.1$, $L = 1001$, $\tau = 0.0001$, $m = 50000$. For better visibility the probability for $x > 50.5$ is normalized for times $t = 40, 45, 50$.

momenta. As a result, the mean value of the wave packet's momentum increases after tunneling through the barrier. Therefore, one would expect that the center of the wave packet gains speed after tunneling.
In general the tunneling of the wave packet can be seen in our result. The wave packet impinges on the barrier and is reflected and transmitted with certain probabilities. As in the case without a barrier, the spread of the probability increases as time evolves. The effect of gained speed after tunneling can be seen in figure 3. The probabilities at times $t = 40, 45, 50$ in the case of a potential barrier (continuous) and no potential barrier (dashed) are compared. One can see,

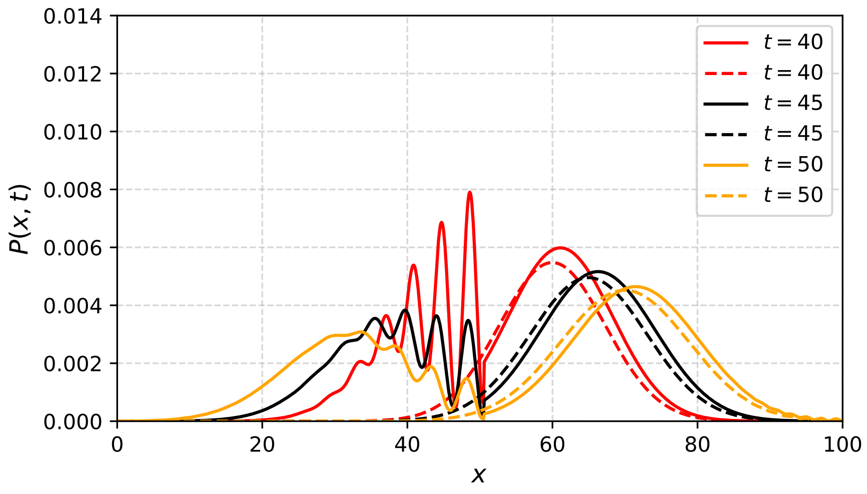

Figure 3: Simulated probability $P(x,t)$ at different times $t = 40, 45, 50$ in the case of a potential barrier (continuous) and no potential barrier (dashed). The effect of gained speed in the case of the barrier is clearly visible.

that at all three times the tunneled wave packet is already further evolved in space than the wave packet in free space.

4. Discussion

The TDSE in 1D was succesfully solved for different cases using the `Second-order product formula algorithm`. It was shown that the spread in space of the wave packet increases over time in both cases. Secondly, the tunneling effect was observed. Furthermore, we were able to confirm an increased speed of the wave packet after tunneling through a potential barrier compared to the wave packet evolving in free space. In general, our simulations agree with the theory quite well.

A. Code used for simulation and animation

```python
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation, PillowWriter
import h5py as h5

# simulation parameters
sigma = 3.0
x_0 = 20.0
q = 1.0
Delta = 0.1
L = 1001
tau = 0.001
m = 50000

###### Change parameters here ####
barrier = False  # whether simulating with barier
simulation = False  # whether new simualtion data should be produced
animate = False # whether simulation data should be animated
plotting = False # whether simualtion data should be plotted
compare_plot = True # whether compare plot should be performed
P_norm_thres = 0.0001 # in case of a barrier, probability distribution
                      # is only normalized once probablity at grid point at
                      # the right of the barrier surpasses this value

######

x_arr = np.linspace(0., (L-1)*Delta, num=L)
time_arr = tau * np.arange(0,m+1)

# building potential in simulation box
barrier_xleft = 50.0
barrier_xright = 50.5
index_left = int(barrier_xleft / Delta)
index_right = int(barrier_xright / Delta)
V_arr = np.zeros(L)

if barrier:
    V_arr[index_left: index_right + 1] = 2
    extra_string = 'barrierNorm'
else:
    extra_string = 'NObarrier'

# calculateing approximate time-evolution operator U
V_diag_0 = np.exp(-1j * (tau * (1 + Delta**2 * V_arr) / Delta**2))
exp_V = np.diag(V_diag_0)

c = np.cos(tau  / (4 * Delta**2))
s = np.sin(tau  / (4 * Delta**2))

K1_diag_0 = np.full(L, c)
K1_diag_0[-1] = 1
K1_diag_1 = np.full(L-1, 1j * s)
K1_diag_1[1:L:2] = 0
exp_K1 = np.diag(K1_diag_1, -1) + np.diag(K1_diag_0, 0) + np.diag(K1_diag_1, 1)

K2_diag_0 = np.full(L, c)
K2_diag_0[0] = 1
K2_diag_1 = np.full(L-1, 1j * s)
K2_diag_1[0:L:2] = 0
exp_K2 = np.diag(K2_diag_1, -1) + np.diag(K2_diag_0, 0) + np.diag(K2_diag_1, 1)
```

1

```python
U = (exp_K1 @ exp_K2 @ exp_V @ exp_K2 @ exp_K1)

# Initialize Phi
Phi = ((2 * np.pi * sigma**2)**(-1 / 4) * np.exp(1j * (x_arr - x_0)) *
        np.exp(-(x_arr - x_0)**2 / (4 * sigma**2)))


# setting up storage frame
storage_frame = 500
storage_steps = int(m / storage_frame) + 1

# storage path
path = './TDSE' + extra_string + '.hdf5'

# excecuting the actual simulation
##################################
if simulation:

    with h5.File(path, "a") as file:
        # setting up the storage file of type 'hdf5'
        file.create_dataset("t", shape=(storage_steps,), maxshape=(None,),
                            chunks=True, dtype=float)
        file.create_dataset("P", shape=(storage_steps, L), maxshape=(None, L),
                            chunks=True, dtype=float)

        k_storage = 0
        count = 0
        normalize = False

        # actual simulation loop
        for step in range(0, m+1):

            # application of approximate operator U
            Phi = U @ Phi

            if step % 2000 == 0:
                print('step:', str(step))

            if (step % storage_frame) == 0:
                # calculating probability
                P = np.absolute(Phi)**2 * Delta

                # Normalizing if x > 50.5
                # and P[index_right+1] > P_norm_thres

                if barrier:
                    P_barrier_right = P[index_right+1]

                    if (count == 0) and P_barrier_right > P_norm_thres:
                        normalize = True
                        count += 1

                    if normalize:
                        #print('I am normalizing:')
                        norm_fac = np.sum(P[index_right+1:])
                        #print('norm_fac', norm_fac)
                        P[index_right+1:] = P[index_right+1:] / norm_fac


                file["t"][k_storage] = time_arr[step]
                file["P"][k_storage] = P
                k_storage += 1


def animate_P(path, extra_string=""):
    """
    Generates a gif displaying the solution for the probability that was
    simulated with the product formula algorithm
    """
    fig, ax = plt.subplots(figsize=(5,3.5))
    fig.set_tight_layout(True)


    lineP, = ax.plot(x_arr, np.zeros_like(x_arr), linewidth=0.75)

    def update(i):
        title = 'time={0:.1f}'.format(t[i])
        lineP.set_ydata(P[i])
        ax.set_title(title, fontsize=12)

        return (lineP, ax)

    data = h5.File(path, "r")

    t = data["t"][:]
    P = data["P"][:]
    data.close()

    if barrier:
        ax.axvspan(x_arr[index_left], x_arr[index_right], color='green',
                   alpha=0.3, label='barrier')
    ax.set_xlim(0,x_arr[-1])
    ax.set_ylim(0, 0.014)
    ax.set_xlabel("$x$", fontsize=12)
    ax.set_ylabel("$P(x,t)$", fontsize=12)
    ax.grid(linestyle="--", alpha=0.5)
    ax.legend(loc=1, fontsize=10)

    anim = FuncAnimation(fig, update, frames=np.arange(t.shape[0]))

    anim.save('{0}.gif'.format(extra_string), writer=PillowWriter(fps=60))
    plt.show()

if animate:
    animate_P(path, extra_string)


def plot_func(path, extra_string=''):
    """
    Generates plot of probabilities at times t=0.0, 5.0, 40.0, 45.0, 50.0
    """

    data = h5.File(path, "r")

    t = data["t"][:]
    P = data["P"][:]
    data.close()

    times_plot = [0.0, 5.0, 40.0, 45.0, 50.0]
    colors = ['b', 'g', 'r', 'k', 'orange']
    fig, ax = plt.subplots(figsize=(6,3.5))
    if barrier:
        ax.axvspan(x_arr[index_left], x_arr[index_right], color='green',
                   alpha=0.3, label='barrier')
    ax.set_xlim(0,x_arr[-1])
    ax.set_ylim(0, 0.014)
    ax.set_xlabel('$x$', fontsize=12)
    ax.set_ylabel('$P(x,t)$', fontsize=12)
    ax.grid(linestyle='--', alpha=0.5)

    for i, times in enumerate(times_plot):

        index_plot = np.where(t == times)[0][0]
        ax.plot(x_arr, P[index_plot], color=colors[i],
                label='$t={0:.0f}$'.format(times))

    ax.legend(fontsize=10)
    fig.tight_layout()
    fig.savefig('{0}.pdf'.format(extra_string))

if plotting:
    plot_func(path, extra_string)

def compare_plot():
    """
    Generates plot of probabilities at times t=40.0, 45.0, 50.0 for both cases
    in order to visualize increased speed of tunneled wave packet
    """
    path_bar = './TDSEbarrierNorm.hdf5'
    path_NObar = './TDSENObarrier.hdf5'

    data_bar = h5.File(path_bar, "r")
    t_bar = data_bar["t"][:]
    P_bar = data_bar["P"][:]
    data_bar.close()

    data_NObar = h5.File(path_NObar, "r")
    t_NObar = data_NObar["t"][:]
    P_NObar = data_NObar["P"][:]
    data_NObar.close()

    times_plot = [40.0, 45.0, 50.0]
    colors = ['r', 'k', 'orange']

    fig, ax = plt.subplots(figsize=(6,3.5))
    ax.set_xlim(0,x_arr[-1])
    ax.set_ylim(0, 0.014)
    ax.set_xlabel('$x$', fontsize=12)
    ax.set_ylabel('$P(x,t)$', fontsize=12)
    ax.grid(linestyle='--', alpha=0.5)

    for i, times in enumerate(times_plot):

        index_plot_bar = np.where(t_bar == times)[0][0]
        index_plot_NObar = np.where(t_NObar == times)[0][0]
        ax.plot(x_arr, P_bar[index_plot_bar], color=colors[i],
                label='$t={0:.0f}$'.format(times))
        ax.plot(x_arr, P_NObar[index_plot_NObar],'--', color=colors[i],
                label='$t={0:.0f}$'.format(times))
    ax.legend(fontsize=10)
    fig.tight_layout()
    fig.savefig('compare.pdf')

if compare_plot:
    compare_plot()
```