# Computational Physics - Exercise 6

Joel Schumacher, 309231, joel.schumacher@rwth-aachen.de

July 7, 2017

# Time-dependent Schrödinger Equation

## Introduction

In this exercise the goal is to solve the time-dependent Schrödinger equation in one dimension:

$$i\hbar\frac{\partial}{\partial t}\Psi(x,t) = \left(-\frac{\hbar^2}{2M}\frac{\partial}{\partial x^2} + V(x)\right)\Psi(x,t)$$

with $\hbar = M = 1$ and

$$V(x) = \begin{cases} 2 & 50 \leq x \leq 50.5 \\ 0 & otherwise \end{cases}$$

and with boundary conditions:

$$\Psi(x, t=0) = \frac{1}{(2\pi\sigma^2)^{1/4}}\exp\left(iq(x-x_0)\right)\exp\left(-\frac{(x-x_0)^2}{4\sigma^2}\right)$$

with $\sigma = 3$, $x_0 = 20$, $q = 1$, which represents a gaussian wave packet at time $t = 0$.

## Simulation model

Using finite differences the spatial derivative in the Schrödinger equation becomes:

$$\frac{\partial^2}{\partial x^2}\Psi(x,t) = \frac{\Psi(x+\Delta,t) - 2\Psi(x,t) + \Psi(x-\Delta,t)}{\Delta^2}$$

with $\Delta = 0.1$ being the spacing of our simulation grid.
The Schrödinger equation in one dimension then becomes

$$\begin{aligned} i\frac{\partial}{\partial t}\Psi(x,t) &= -\frac{\Psi(x+\Delta,t) - 2\Psi(x,t) + \Psi(x-\Delta,t)}{2\Delta^2} + V(x)\Psi(x,t) \\ &= \frac{1}{\Delta^2}\left(-\frac{1}{2}\Psi(x+\Delta,t) - \frac{1}{2}\Psi(x-\Delta,t) + \left(1 + \Delta^2 V(x)\right)\Psi(x,t)\right). \end{aligned}$$

Our discretized Hamiltonian can then be read as:

$$i\frac{\partial}{\partial t}\begin{pmatrix}\Psi_1(t)\\ \Psi_2(t)\\ \Psi_3(t)\\ \vdots\\ \vdots\\ \Psi_L(t)\end{pmatrix} = \underbrace{\frac{1}{\Delta^2}\begin{pmatrix} 1+\Delta^2 V_1 & -1/2 & 0 & & & 0 \\ -1/2 & 1+\Delta^2 V_2 & -1/2 & 0 & & 0 \\ 0 & -1/2 & 1+\Delta^2 V_3 & & & \\ & & & \ddots & & 0 \\ & & & & & -1/2 \\ 0 & & & 0 & -1/2 & 1+\Delta^2 V_L \end{pmatrix}}_{H}\begin{pmatrix}\Psi_1(t)\\ \Psi_2(t)\\ \Psi_3(t)\\ \vdots\\ \vdots\\ \Psi_L(t)\end{pmatrix}$$

with $V_i = V(i\Delta)$, $\Psi_i(t) = \Psi(x = i\Delta, t)$ and $L = 1001$ being the size of our simulation grid (in number of grid points).

The Hamiltonian can then be decomposed into a sum:

$$H = V + K_1 + K_2 = \frac{1}{\Delta^2}\begin{pmatrix} 1+\Delta^2 V_1 & 0 & 0 & & & 0 \\ 0 & 1+\Delta^2 V_2 & 0 & 0 & & 0 \\ 0 & 0 & 1+\Delta^2 V_3 & & & \\ & & & \ddots & & 0 \\ & & & & 1+\Delta^2 V_{L-1} & 0 \\ 0 & & & 0 & 0 & 1+\Delta^2 V_L \end{pmatrix}$$

$$+ \frac{1}{\Delta^2}\begin{pmatrix} 0 & -1/2 & 0 & & & 0 \\ -1/2 & 0 & 0 & & & \\ 0 & 0 & 0 & & & \\ & & & \ddots & -1/2 & 0 \\ & & & -1/2 & 0 & 0 \\ 0 & & & 0 & 0 & 0 \end{pmatrix} + \frac{1}{\Delta^2}\begin{pmatrix} 0 & 0 & 0 & & & 0 \\ 0 & 0 & -1/2 & & & \\ 0 & -1/2 & 0 & & & \\ & & & \ddots & & 0 \\ & & & & 0 & -1/2 \\ 0 & & & 0 & -1/2 & 0 \end{pmatrix}$$

The time evolution operator can then be written as:

$$U(\tau) = e^{-i\tau H} \approx e^{-i\tau K_1/2} e^{-i\tau K_2/2} e^{-i\tau V} e^{-i\tau K_2/2} e^{-i\tau K_1/2}$$

where $\tau = 0.001$ is the time step of our simulation.

The full expressions are the following:

$$e^{-i\tau K_1/2} = \begin{pmatrix} c & is & & & & & \dots & 0 \\ is & c & 0 & & & & & \vdots \\ & 0 & c & is & & & & \\ & & is & c & 0 & & & \\ & & & 0 & c & is & & \\ & & & & is & c & 0 & \\ & & & & & 0 & \ddots & is \\ \vdots & & & & & & is & \ddots & 0 \\ 0 & \dots & & & & & 0 & 1 \end{pmatrix}$$

$$e^{-i\tau K_2/2} = \begin{pmatrix} 1 & 0 & & & & & \dots & 0 \\ 0 & c & is & & & & & \vdots \\ & is & c & 0 & & & & \\ & & 0 & c & is & & & \\ & & & is & c & 0 & & \\ & & & & 0 & c & is & \\ & & & & & is & \ddots & 0 \\ \vdots & & & & & & 0 & \ddots & is \\ 0 & \dots & & & & & is & c \end{pmatrix}$$

$$e^{-i\tau V} = \frac{1}{\Delta^2}\begin{pmatrix} 1+\Delta^2 V_1 & 0 & 0 & & & & & 0 \\ 0 & 1+\Delta^2 V_2 & 0 & 0 & & & & 0 \\ 0 & 0 & 1+\Delta^2 V_3 & & & & & \\ & & & \ddots & & & & 0 \\ & & & & 1+\Delta^2 V_{L-1} & & & 0 \\ & & & & & 0 & 0 & 1+\Delta^2 V_L \end{pmatrix}$$

where $c = \cos\left(\tau/(4\Delta^2)\right)$ and $s = \sin\left(\tau/(4\Delta^2)\right)$.

In each timestep a set of complex values for each grid point is then updated using the time evolution operator.

## Simulation results

The time evolution of the probability distribution of the wave function $P(x,t) = |\Psi(x,t)|^2$ is shown in figure 1. In figure 2 a comparison of the cases with a potential barrier and without a potential barrier can be seen. In the case of the potential barrier the probability of the transmitted portion has been rescaled, so that the distribution with the maximum transmission has a total probability of 1 behind the barrier for easier comparison of the two cases. It is visibile that this rescaling factor is $\sim 3$, implying that the total transmission probability is about 30%. It can also be observed that the wave packets move faster after the transmission if a barrier is present compared to the case of no barrier.

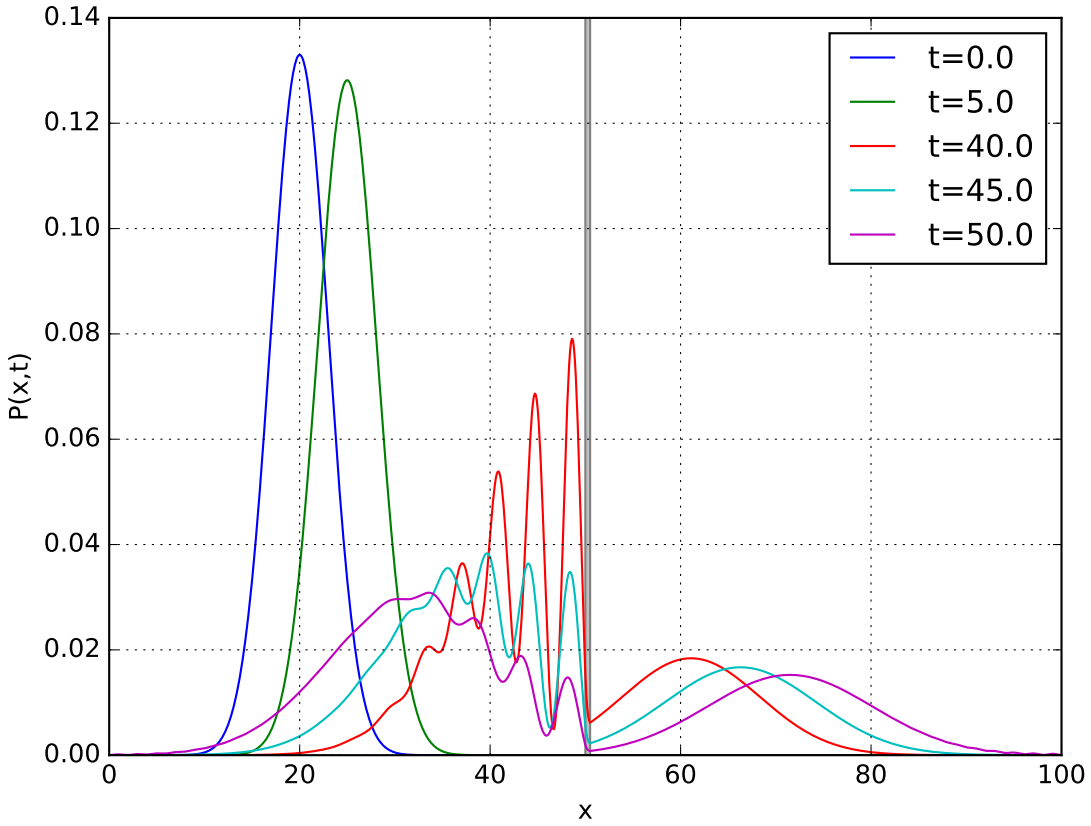


Figure 1: The probability distribution of the wave packet at different times. The barrier is represented by the grey box.

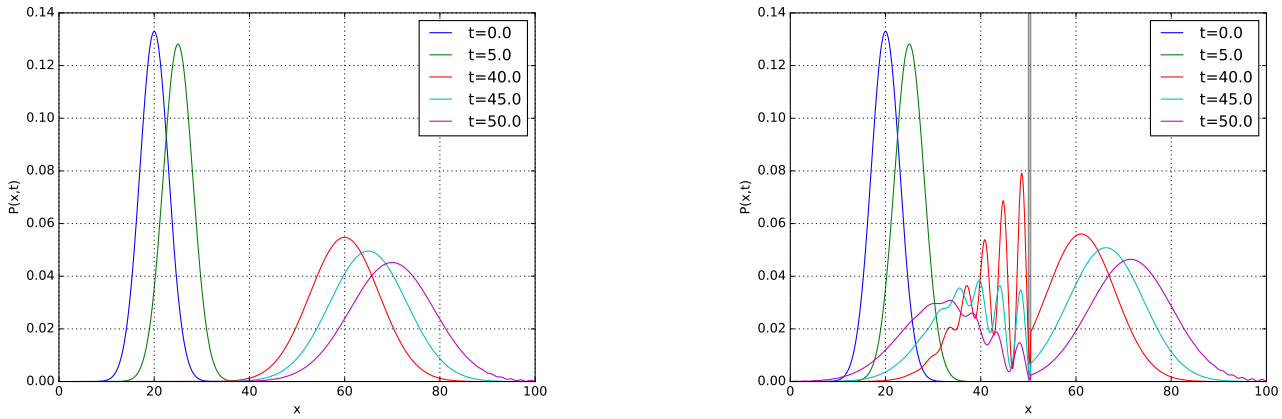


Figure 2: The probability distribution of the wave packet at different times. Without a potential barrier on the left and with a potential barrier on the right. The right plot has been rescaled, so that probability distribution with the most transmission has total probability 1 on the right side of the barrier.

## Discussion

The simulation seems to give a fairly correct result, that matches with the expected solution of the Schrödinger equation in that the wave is partially transmitted by a small fraction and the wave packet gaining speed after the transmission. This phenomenon can be explained by either a simple, intuitive picture of a Ball tunneling through the tip of a potential hill and rolling down the rest or, more accurately, the transmission probability depending on the energy of the quantum, with higher energies being more probable. This potential barrier therefore effectively acts as a high-pass filter, favoring the components of the wave packet with higher momentum. The fact that the wave packet is comprised of different momenta is also visible in the dispersion of the packet, which is also very much in line with the expectation.

In conclusion the product formula approach is an effective and simple approach for solving the one-dimensional Schrödinger equation. This approach can not be trivially extended to higher dimension, since mapping a e.g. 2 dimensional grid of points to a 1-dimensional vector will result in a differently structured matrix, which complicates the structure of the exponential of $K_1$ and $K_2$. Also lowering the time step size introduces significant oscillations, which can still be observed in the plots for $t = 50$ in the vicinity of $x = 100$, meaning the time step has to be sufficiently low for accurate results.

## Appendix

```
1  import math
2  import cmath
3  import numpy as np
4  import matplotlib.pyplot as plt
5
6  sigma = 3
7  x0 = 20
8  q = 1
9
10 Delta = 0.1
11 L = 1001
12 tau = 0.001
13 m = 50000
14
15 rescale = True
```

```
#rescale = False
V_barrier = 2
#V_barrier = 0

def V(x):
    if x >= 50.0 and x <= 50.5:
        return V_barrier
    else:
        return 0

def psi_t0(x):
    ret = math.pow(2 * math.pi * sigma*sigma, -0.25)
    ret *= cmath.exp(1j * q * (x-x0))
    ret *= math.exp(-(x-x0)*(x-x0)/(4*sigma*sigma))
    return ret

def pd(psi):
    v = np.absolute(psi)
    return v*v

psi = np.zeros(L, dtype=complex)
for l in range(L):
    psi[l] = psi_t0(l*Delta)

expV = np.zeros((L,L), dtype=complex)
for l in range(L):
    expV[l][l] = cmath.exp(-1j * tau * V(l*Delta))

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

expK1 = np.eye(L, dtype=complex) * c
expK1[L-1][L-1] = 1
expK2 = np.eye(L, dtype=complex) * c
expK2[1][1] = 1

for l in range(L):
    if l+1 < L:
        if l % 2 == 0:
            expK1[l+1][l] = expK1[l][l+1] = 1j * s
        else:
            expK2[l+1][l] = expK2[l][l+1] = 1j * s

U = expK1 @ expK2 @ expV @ expK2 @ expK1

x = np.arange(L) * Delta
plt.xlabel("x")
plt.ylabel("P(x,t)")
plt.grid(True)

ax = plt.gca()
ylim = 0.14
plt.ylim(0, ylim)
if V_barrier > 0:
    ax.fill_between([50.0, 50.5], 0, ylim, facecolor='grey', alpha=0.5)

```

```
right_index = math.floor(50.5 / Delta + 0.5)
right_normalization = 0
snapshots = []
for i in range(m+1):
    if math.floor(i/m*100) != math.floor((i-1)/m*100):
        print(math.floor(i/m*100), "%")
    t = i * tau

    # snapshot
    if i in [0, m/10, m/5*4, m/50*45, m]:
        total_prob = np.sum(pd(psi)) * Delta
        right_prob = np.sum(pd(psi[right_index:])) * Delta
        if right_prob > right_normalization:
            right_normalization = right_prob
        snapshots.append((t, pd(psi)))

    psi = np.dot(U, psi)

for t, pdpsi in snapshots:
    if rescale:
        pdpsi[right_index:] /= right_normalization
    plt.plot(x, pdpsi, label="t={}".format(t))

plt.legend()
plt.savefig("tdse_V={}_rescale_right={}.pdf".format(V_barrier, rescale))
plt.show()
```