Report 8: A Suzuki-Trotter product formula approach to solve the time-dependent Schrödinger equation

Don Winter[1] **and Madhulan Suresh**[2]

[1]don.winter@rwth-aachen.de, Mat. No. 431380
[2]madhulan.suresh@rwth-aachen.de, Mat. No. 429481

Introduction

In this report we develop a method to numerically solve the time-dependent Schrödinger equation approximately for free space and barrier potentials. Similar to our last report, we employ the second order Suzuki-Trotter product formula to find an efficient algorithm to update the state vector by sequentially applying partial time-step operators. We analyse the simulation results for the two types of potentials and afterwards explain why the state evolves faster after tunneling through the barrier by considering the Fourier transform of the position wave function.

Simulation model and method

Quantum theory is a probabilistic theory describing the evolution and interaction of systems at the atomic and subatomic scale, so called *quantum* systems. The *state* of a system often called state vector $|\psi\rangle$ is modelled as a complex-valued *ket*-vector in an infinite dimensional Hilbert space which evolves in time according to the time-dependent Schrödinger equation

$$i\hbar\frac{\partial}{\partial t}|\psi\rangle = \hat{H}|\psi\rangle, \tag{1}$$

for a given system Hamiltonian $\hat{H}$. In this report we solve the equation for a particle moving along one dimension subjected to a certain potential V. For this case the Hamiltonian is known and the Schrödinger equation reads

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

We choose units such that $\hbar = m = 1$ and discretize the equation centrally in space by using $x \to j\Delta$ for a small Δ. We obtain

$$i\hbar\frac{\partial}{\partial t}\psi(x,t) = -\frac{\psi(x+\Delta,t) - 2\psi(x,t) + \psi(x-\Delta,t)}{2\Delta^2} + V(x)\psi(x,t) = -\frac{\psi(x+\Delta,t) + \psi(x-\Delta,t)}{2\Delta^2} + \big(\frac{1}{\Delta^2} + V(x)\big)\psi(x,t), \tag{3}$$

which we can rewrite in matrix form by expressing $\psi(x,t)$ as a vector with components defined at discrete space positions $j\Delta$, i.e. $\psi_j(t) = \psi(j\Delta,t)$ for integers $j = 1,2,...,L$.

$$i\frac{\partial}{\partial t}\begin{bmatrix}\psi_1(t)\\ \psi_2(t)\\ \psi_3(t)\\ \vdots\\ \psi_{L-1}(t)\\ \psi_L(t)\end{bmatrix} = \frac{1}{\Delta^2}\begin{bmatrix}1+\Delta^2V_1 & -1/2 & 0 & & & 0\\ -1/2 & 1+\Delta^2V_2 & -1/2 & & & \\ 0 & -1/2 & 1+\Delta^2V_3 & & & \\ & & & \ddots & & 0\\ & & & & 1+\Delta^2V_{L-1} & -1/2\\ 0 & & & 0 & -1/2 & 1+\Delta^2V_L\end{bmatrix}\begin{bmatrix}\psi_1(t)\\ \psi_2(t)\\ \psi_3(t)\\ \vdots\\ \psi_{L-1}(t)\\ \psi_L(t)\end{bmatrix} = \frac{1}{\Delta^2}\hat{H}\begin{bmatrix}\psi_1(t)\\ \psi_2(t)\\ \psi_3(t)\\ \vdots\\ \psi_{L-1}(t)\\ \psi_L(t)\end{bmatrix}, \tag{4}$$

where $V_j = V(j\Delta)$, accordingly. We note that the discretized Hamiltonian $\hat{H}$ is a (real) Hermitian matrix and that the general solution to this equation is

$$\vec{\psi}(t) = e^{-it\hat{H}}\vec{\psi}(t=0) = \hat{U}\vec{\psi}(t=0), \tag{5}$$

yielding the unitary time evolution matrix $\hat{U}$. Due to its unitarity, the norm of $\vec{\psi}(t)$ stays invariant under transformation, which coincides with the conservation of probability at all times in quantum theory. At this point we could already trotterize $\hat{U}$ and run our simulation. However, due to the special structure of $\hat{H}$ a more efficient strategy is to first decompose it into other Hermitian

matrices and then use the approximation $e^{\hat{A}+\hat{B}} \approx e^{\hat{A}} e^{\hat{B}}$. By this decomposition we still assure that the time step operator remains unitary and that our simulation stays unconditionally stable. In the following we will see that this approach furthermore allows us to update the state more efficiently. We decompose the Hamiltonian as follows:

$$\hat{H} = \hat{V} + \hat{K}_1 + \hat{K}_2 = \frac{1}{\Delta^2} \begin{bmatrix} 1+\Delta^2 V_1 & 0 & 0 & & & & 0 \\ 0 & 1+\Delta^2 V_2 & 0 & & & & \\ 0 & 0 & 1+\Delta^2 V_3 & & & & \\ & & & \ddots & & & 0 \\ & & & & 1+\Delta^2 V_{L-1} & 0 & \\ 0 & & & 0 & 0 & 1+\Delta^2 V_L & \end{bmatrix} + \\ + \frac{1}{\Delta^2} \begin{bmatrix} 0 & -1/2 & 0 & & & 0 \\ -1/2 & 0 & 0 & & & \\ 0 & 0 & 0 & & & \\ & & & \ddots & -1/2 & 0 \\ & & & -1/2 & 0 & 0 \\ 0 & & & 0 & 0 & 0 \end{bmatrix} + \frac{1}{\Delta^2} \begin{bmatrix} 0 & 0 & 0 & & & 0 \\ 0 & 0 & -1/2 & & & \\ 0 & -1/2 & 0 & & & \\ & & & \ddots & & 0 \\ & & & & 0 & -1/2 \\ 0 & & & & -1/2 & 0 \end{bmatrix}. \tag{6}$$

We note that $\hat{V}$ is diagonal and $\hat{K}_1$ and $\hat{K}_2$ are (mostly) block-diagonal. Now, we use the second order (symmetrized) product formula to approximate

$$e^{-it\hat{H}} = e^{-it(\hat{V}+\hat{K}_1+\hat{K}_2)} \approx e^{-it\hat{K}_1/2} e^{-it\hat{K}_2/2} e^{-it\hat{V}} e^{-it\hat{K}_2/2} e^{-it\hat{K}_1/2}, \tag{7}$$

which we subsequently trotterize to

$$e^{-it\hat{K}_1/2} e^{-it\hat{K}_2/2} e^{-it\hat{V}} e^{-it\hat{K}_2/2} e^{-it\hat{K}_1/2} \approx \left(e^{-i\tau\hat{K}_1/2} e^{-i\tau\hat{K}_2/2} e^{-i\tau\hat{V}} e^{-i\tau\hat{K}_2/2} e^{-i\tau\hat{K}_1/2}\right)^m \tag{8}$$

with time-step $\tau = t/m$ for large Trotter number m. When we compute the matrix exponentials of the blocks in $\hat{K}_1$ and $\hat{K}_2$ by expanding into exponential series we obtain

$$e^{i\alpha\hat{X}} = \begin{bmatrix} \cos(\alpha) & i\sin(\alpha) \\ i\sin(\alpha) & \cos(\alpha) \end{bmatrix}, \tag{9}$$

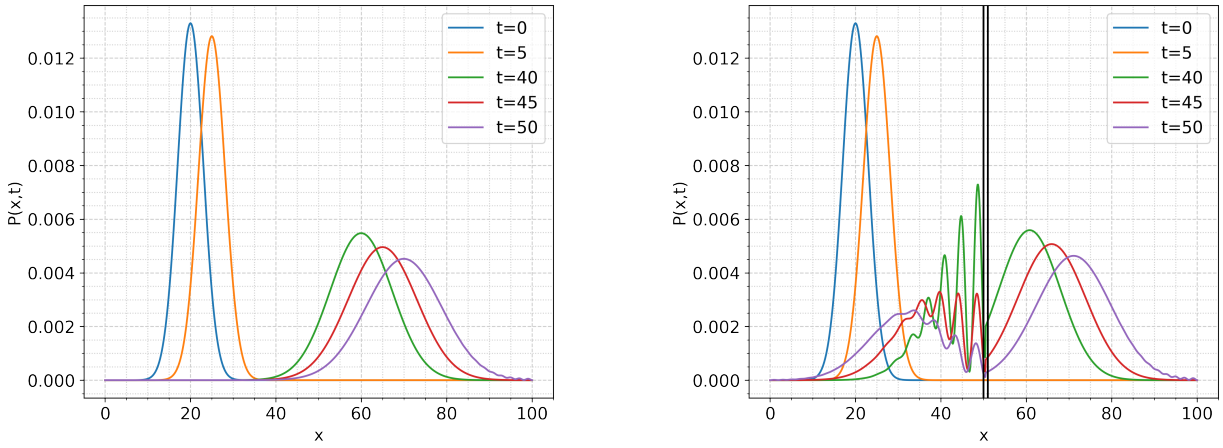
for σ_x-Pauli matrix $\hat{X}$ and $\alpha = \tau/4\Delta^2$. For the 0 in the first (last) position of the block-diagonal matrices we obtain the exponential $e^0 = 1$. Thus, in summary, for each time step $i = 0, ..., m$ we update our discretized state vector $\vec{\psi}$ by the trotterized sequence of matrix exponentials for which we make use of the block-diagonal structure in updating neighboring pairs of vector coefficients by $e^{i\alpha\hat{X}}$ and for the matrix exponential of the potential directly take the Hadamard product. By this procedure we obtain an algorithm which scales linearly in simulation space, i.e. which has complexity $\mathcal{O}(L)$.

Simulation results

For the simulation we use a one-dimensional grid of $L = 1001$ positions with a spacing of $\Delta = 0.1$. We run our simulation for $m = 50000$ time steps of $\tau = 0.001$ by which we achieve a time evolution from $t = 0, ..., 50$. Moreover, we simulate for two initial conditions. First with $V(x) = const. = 0$ and second for $V(50 \leq x \leq 50.5) = 2$ and otherwise 0, i.e. for a potential barrier in the middle of our simulation space. Finally, we initialize the state vector to a Gaussian wave packet

$$\psi(x, t=0) = \frac{1}{\sqrt{\sqrt{2\pi}\sigma}} e^{iq(x-x_0) - (x-x_0)^2/4\sigma^2}, \tag{10}$$

for which we choose $\sigma = 3$, $x_0 = 20$ and $q = 1$. This will produce a wave packet traveling from $x = 20$ to $x = 70$ within the time of the simulation. As we defined simulation space in units of Δ we converted all parameters to our simulation space accordingly. In Fig. 1 we see the results of the simulation. In the case of no potential the probability distribution travels to the right while decreasing in height and increasing in width. During the full evolution the distribution stays Gaussian and the peak moves with a constant velocity. This is the same behavior we get for a classical ensemble (for example water) under time-evolution which was localized at a position x_0 at time $t = 0$. The quantum behavior of our state can be seen in Fig. 1b, in which a large but thin potential barrier is placed in the center of the simulation space. We see that for the first few time steps the state evolves classically, however once it comes into contact with the barrier interference effects take place which produce



(a) Using potential $V_1 = const. = 0$

(b) Using potential V_2-barrier potential

Figure 1. Snapshots of the probability distribution for different time steps. One can see the particle traveling to the right from $x = 20$ to $x = 70$. For V_2-potential the part of the distribution to the right of the potential barrier (i.e. the part that *tunneled* through the barrier) has been rescaled by by the maximum total probability in this region.

highly non-classical probabilities. Thus, we observe the state tunneling through the barrier with a certain probability and evolving classically again to the right of the barrier. Another interesting observation we can make is that the wave function to the right of the barrier, after tunneling, travels faster than for the case without barrier. To illustrate this we can simply superpose the two plots of Fig. 1 into a single one (Fig. 2). We observe that the probabilities for V_2 to the left of the barrier line up with

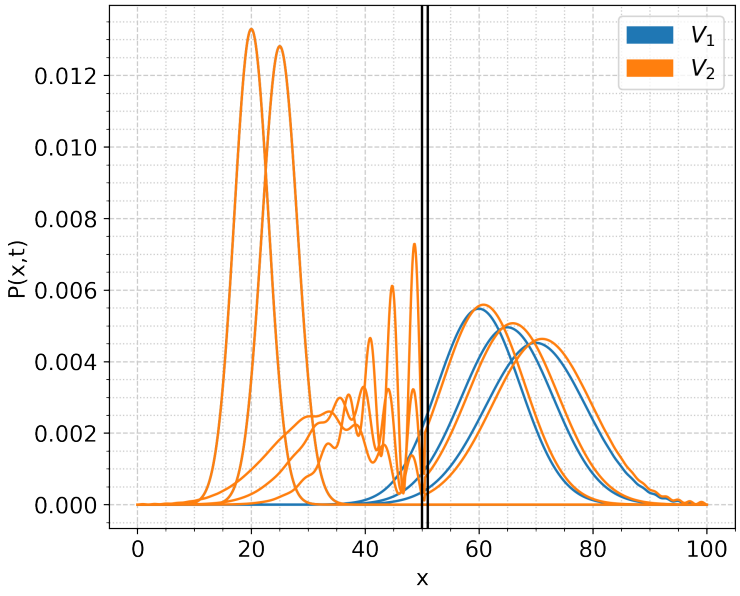
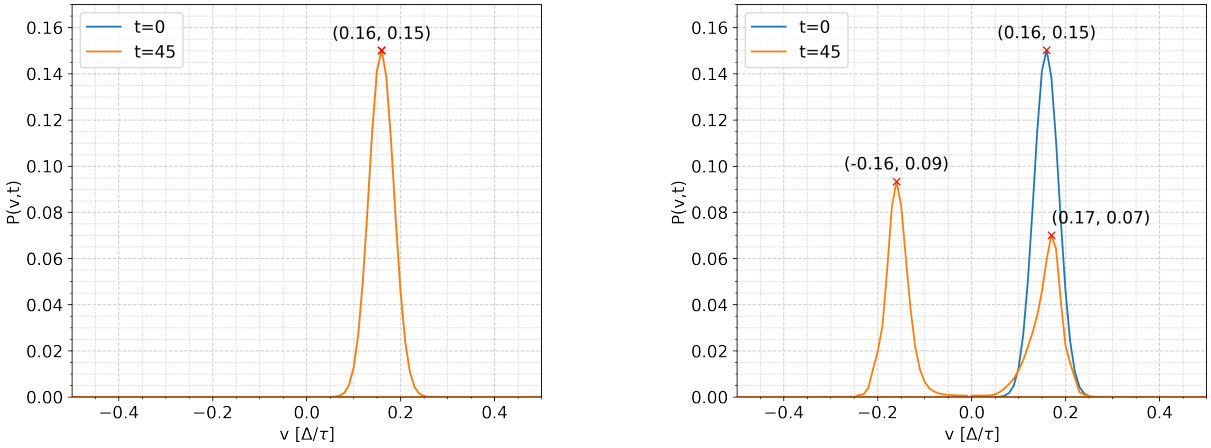


Figure 2. Comparison of time evolved probability distributions under the influence of two different potentials V_1 and V_2.

the ones for V_1, while on the right they are slightly ahead of these. This curious behavior happens as our initial wave function is a Gaussian wave packet, i.e. a superposition of plane waves of different wave vectors and the transmission coefficient, i.e. the

tunnel rate is an exponential function of energy of an incident wave, which itself is related to the square of the momentum and thus to the wave vector. As a consequence, there is a much larger probability for the plane waves of large wave vectors to be transmitted through the barrier and a very low probability for the respective lower energetic ones. The barrier acts like a high-pass filter and therefore the tunneled fraction of the wave function evolves faster after tunneling. One way to show this is by taking the discrete Fourier transform of the position wave function at a time step before and after tunneling. This transforms the wave function to momentum representation, or in our case velocity and wave vector representation as $\hbar = m = 1$. The result we can see in Fig. 3. We can see that in the case of the barrier the transmitted part gained in momentum, as not only the peak but also the whole distribution is *slanted* to the right, while the reflected part is a Gaussian with additional low momentum contributions and negative momentum, i.e. travelling to the left.



(a) Using potential $V_1 = const. = 0$

(b) Using potential V_2-barrier potential

Figure 3. Probability distribution of state in momentum representation. For the case of no potential the distribution is fixed in time. For the case with potential barrier we see at $t = 45$, after encountering the barrier the state splits up into a reflected part and a transmitted part. The maximum of the transmitted part has a slightly *slanted* distribution with a maximum at a larger momentum value.

Discussion

In summary, we simulated the time evolution of a Gaussian wave packet in free space and through a potential barrier by using the time-dependent Schrödinger equation in a product formula simulation approach. We first illustrated how we derived our simulation algorithm from the Schrödinger equation, then discussed the setup of our simulation space and finally presented our results. In the end we tried to illustrate in more detail why the wave packet, after tunneling through the barrier has increased in speed, i.e. group velocity. For this means we Fourier transformed the wave function into momentum representation and argued by considering the probability distribution of the momentum spectrum.

Appendix

Common libraries and parameters

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib
font = {'size': 14}
matplotlib.rc('font', **font)

# discretization parameters
delta = 0.1
L = 1001
tau = 0.001
m = 50_000
```

```

# 2x2 time-step block
X = np.array([[0,1],[1,0]])
I = np.eye(2)
a = tau/(4*delta**2)
A = np.cos(a)*I + 1j*np.sin(a)*X

# given parameters in l-space
l = np.array(range(L))
l_0 = 20/delta
q = 1*delta
sigma = 3/delta

# potentials in l-space
V1, V2 = np.zeros(L), np.zeros(L)
V2[int(50/delta):int(50.5/delta)] = 2
Vs = [V1,V2]
```

Simulation

```
from common import *

def run():

    # Amount of 2-element pairs for block-update
    L_half = (L-1) // 2

    # Time steps for snapshots
    T = np.array([0,5,40,45,50])
    Ps = []

    for j, V in enumerate(Vs):

        # initial condition
        psi = np.zeros(L)
        psi = (2*np.pi*sigma**2)**(-1/4)*np.exp(1j*q*(l-l_0))*np.exp(-(l-l_0)**2/(4*sigma**2))

        P = []
        for i in range(m+1):

            # e^(-iTK2/2)e^(-iTK1/2)|psi>
            psi[:-1] = np.hstack([A @ v for v in np.split(psi[:-1], L_half)]).ravel()
            psi[1:] = np.hstack([A @ v for v in np.split(psi[1:], L_half)]).ravel()

            # e^(-iTV)|psi>
            psi = np.exp(-1j*tau*(delta**(-2) + V)) * psi

            # e^(-iTK1/2)e^(-iTK2/2)|psi>
            psi[1:] = np.hstack([A @ v for v in np.split(psi[1:], L_half)]).ravel()
            psi[:-1] = np.hstack([A @ v for v in np.split(psi[:-1], L_half)]).ravel()

            # Save |psi> for snapshot
            if i in T/tau:
                P.append(psi)
        Ps.append(P)

    return np.array(Ps)
```

Probability distribution plots

```
from common import *
import sim

psis = sim.run()
probs = np.zeros_like(psis)

for i in range(len(Vs)):
    for j in range(len(T)):
      # Calculate probabilit from state vector
        probs[i,j] = psis[i,j].real**2 + psis[i,j].imag**2

```

```
# normalize the tunneled portion
max_tot_prob = np.max(np.sum(probs[1,:,int(50.5/delta):], axis=1))
probs[1,:,int(50.5/delta):] = probs[1,:,int(50.5/delta):] / max_tot_prob

# plot probabilities
cols = ['tab:blue','tab:orange']
for i,v_col in enumerate(cols):
    for j,t in enumerate(T):
        plt.plot(x*delta, probs[i,j,:], c=v_col)
```

Momentum distribution plots

```
from common import *
from scipy.fftpack import fft, fftfreq
import sim

psis = sim.run()

for i in range(len(Vs)):
    for j in [0,-2]:

        # Fourier transform state vector
        fx = fftfreq(L,delta)
        fy = fft(psis[i,j])

        # Convert to probability
        p = (fy.real**2 + fy.imag**2)
        p /= np.sum(p) # renormalize

        # Find maximum
        xmax = fx[np.argmax(p)]
        ymax = p.max()

        # Plot transform and maximum
        plt.plot(fx, p, label='t=%d'%T[j])
        plt.plot(xmax, ymax, 'x', c='red');

plt.xlabel(r'v [$\Delta/\tau$]');
plt.ylabel(r'P(v,t)');
```