Report 7: One-dimensional numerical diffusion from Suzuki-Trotter product formula and Random Walk

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

[1]431380, don.winter@rwth-aachen.de
[2]429481, madhulan.suresh@rwth-aachen.de

Introduction

In this exercise, we simulate the physical phenomenon of diffusion in one dimension. In the following we present two different simulation methods, namely a Suzuki-Trotter product formula approach in which we split the diffusion Hamiltonian into a sum of block-diagonal matrices by which we update the state vector and subsequently calculate observables of interest. The second method is a stochastic one-dimensional Random Walk which was already discussed in a previous report, therefore we only briefly discuss it here.

Simulation model and method

Our main focus in this report is the Suzuki-Trotter product formula approach. We start with the analytical model of diffusion, the well-known *Diffusion Equation*, which is the result of Fick's Law (A. Fick, 1855) and the *Continuity Equation*

$$\vec{j}(\vec{r},t) = -D\nabla N(\vec{r},t), \qquad \frac{\partial N(\vec{r},t)}{\partial t} + \vec{\nabla}\cdot\vec{j} = 0 \tag{1}$$

for which we for simplicity consider only one-dimensional diffusion of a substance which diffuses with constant diffusion coefficient D. Combining Eqs. 1, we arrive at the familiar form of the partial differential equation:

$$\frac{\partial N(x,t)}{\partial t} = -D\frac{\partial^2 N(x,t)}{\partial x^2}, \tag{2}$$

for particle density $N(x,t)$ at space position x and time t. If we discretize this equation centrally in space we obtain

$$\frac{\partial N(x,t)}{\partial t} = -D\frac{N(x+\Delta,t) - 2N(x,t) + N(x-\Delta,t)}{\Delta^2}, \tag{3}$$

which we can rewrite as a matrix equation by defining $\vec{N}(t) = N(\vec{x},t)$ the particle density at discrete space location $x_i = \Delta\cdot i$:

$$\frac{\partial \vec{N}(t)}{\partial t} = -\frac{D}{\Delta^2}\begin{bmatrix} -2 & 1 & 0 & 0 & \dots & 0 \\ 1 & -2 & 1 & 0 & \dots & 0 \\ 0 & 1 & -2 & 1 & \dots & 0 \\ 0 & 0 & 1 & -2 & \dots & 0 \\ \vdots & & & & \ddots & \vdots \\ 0 & 0 & 0 & \dots & 1 & -2 \end{bmatrix}\begin{bmatrix} N(x_1,t) \\ N(x_2,t) \\ N(x_3,t) \\ N(x_4,t) \\ \vdots \\ N(x_L,t) \end{bmatrix} = \alpha\hat{H}\vec{N}(t), \tag{4}$$

where $\alpha = -D/\Delta^2$. This is a common first-order differential matrix equation with formal solution $\vec{N}(t) = e^{\alpha t\hat{H}}\vec{N}_0$. Now, by inspection we see that $\hat{H}$ can be decomposed into the sum of two (mostly) block-diagonal matrices $\hat{A}$ and $\hat{B}$ where

$$\hat{A} = \begin{bmatrix} \hat{X} & 0 & \dots & 0 \\ 0 & \hat{X} & \dots & 0 \\ \vdots & & \ddots & \vdots \\ 0 & 0 & \dots & 1 \end{bmatrix}, \qquad \hat{B} = \begin{bmatrix} 1 & 0 & \dots & 0 \\ 0 & \hat{X} & \dots & 0 \\ \vdots & & \ddots & 0 \\ 0 & 0 & \dots & \hat{X} \end{bmatrix} \quad \text{with} \quad \hat{X} = \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}, \tag{5}$$

such that $\hat{H} = \hat{A} + \hat{B}$. Next we approximate the time-evolution from $t = 0$ to $t = T$ in (small) time steps of τ by the second order Suzuki-Trotter product formula:

$$e^{T\alpha(\hat{A}+\hat{B})} = \lim_{m\to\infty}(e^{\alpha\tau\hat{A}/2}e^{\alpha\tau\hat{B}}e^{\alpha\tau\hat{A}/2})^m \approx (e^{\alpha\tau\hat{A}/2}e^{\alpha\tau\hat{B}}e^{\alpha\tau\hat{A}/2})^m, \tag{6}$$

for which we used $\tau = T/m$ and the approximation being valid for large enough Trotter number m. As we know $\hat{A}$ and $\hat{B}$ are block-diagonal we make use of the theorem that matrix exponentiation of block-diagonal matrices are equal to the matrix of the exponents of its blocks. As the only block in both matrices is $\hat{X}$, we only need to calculate its exponential:

$$e^{\tau\alpha\hat{X}} = \frac{1}{2}\begin{bmatrix} 1+e^{2\tau\alpha} & 1-e^{2\tau\alpha} \\ 1-e^{2\tau\alpha} & 1+e^{2\tau\alpha} \end{bmatrix}. \tag{7}$$

Our (density) state update rule thus becomes the sequential matrix multiplication of $e^{\tau\alpha\hat{X}}$ to (changing) neighboring coefficients in $\vec{N}(t)$ according to:

1. $\begin{bmatrix} N(x_i,t) \\ N(x_{i+1},t) \end{bmatrix} \to e^{\tau\alpha\hat{X}/2}\begin{bmatrix} N(x_i,t) \\ N(x_{i+1},t) \end{bmatrix}$ for $i = 1,...,L-1$ and $N(x_L,t) \to e^{\tau\alpha/2}N(x_L,t)$.
2. $\begin{bmatrix} N(x_i,t) \\ N(x_{i+1},t) \end{bmatrix} \to e^{\tau\alpha\hat{X}}\begin{bmatrix} N(x_i,t) \\ N(x_{i+1},t) \end{bmatrix}$ for $i = 2,...,L$ and $N(x_1,t) \to e^{\tau\alpha}N(x_1,t)$.
3. $\begin{bmatrix} N(x_i,t) \\ N(x_{i+1},t) \end{bmatrix} \to e^{\tau\alpha\hat{X}/2}\begin{bmatrix} N(x_i,t) \\ N(x_{i+1},t) \end{bmatrix}$ for $i = 1,...,L-1$ and $N(x_L,t) \to e^{\tau\alpha/2}N(x_L,t)$, again.

After one iteration of this time update we can calculate the pth moment of the particle distribution over the grid with respect to the initial position (i.e. source) x_0 of the particles by

$$\langle x^P(t)\rangle \approx \frac{\sum_{x_i=1}^{L}(x_i-x_0)^P N(x_i,t)}{\sum_{x_i=1}^{L} N(x_i,t)}, \tag{8}$$

Finally, we find the variance about the source position x_0 by

$$Var[x(t)] = \langle x^2(t)\rangle - \langle x(t)\rangle^2, \tag{9}$$

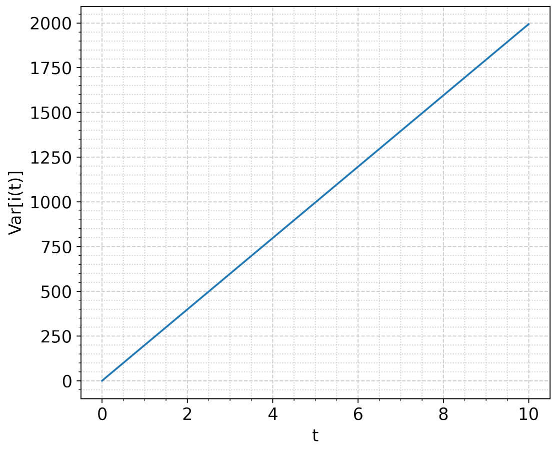
which we can convert to integer grid positions $i = 1,...,L$ by rescaling $Var[x] \to Var[i] = Var[x]/\Delta^2$. This conversion becomes important when we want to compare our results to the Random Walk approach which operates on the discrete grid points i. For the Random Walk we update a vector of length L for every time step τ, s.t. the density at one position is updated by half the amount of the densities evaluated on its neighboring grid points:

$$N(x,t+\tau) = \frac{1}{2}(N(x+\Delta,t)+N(x-\Delta,t)). \tag{10}$$

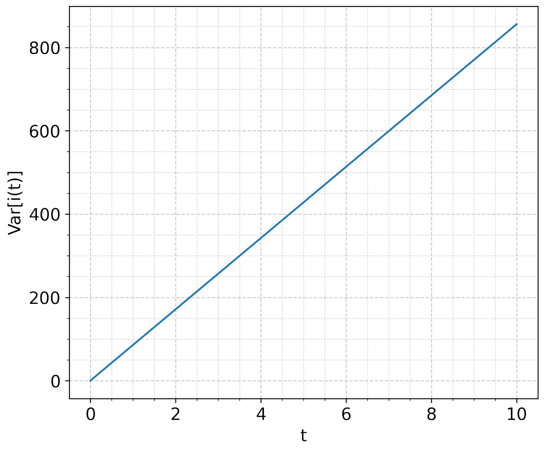
We run the Random Walk simulation with the same parameters as the product formula approach. Next, we take a look at the simulation results and present our implementation.

Results

For the specified parameters of diffusion coefficient $D = 1$, total number of grid points $L = 1001$, grid spacing $\Delta = 0.1$, time step size $\tau = 0.001$ and total number of time steps $m = 10000$ gives a total time evolution from $t = 0,...,10$. Additionally, we considered two initial conditions for all particles starting in the center of the grid (condition 1) and all particles starting at the left side of the grid (condition 2). The resulting plots for both these conditions can be seen in Fig. 1. We notice in both cases a linear increase in spread (variance) over time. For the second initial condition we observe that the slope is by more than a factor of two smaller than for the first condition. When we plot the total density over time we see that the density is leaking out of the system (at the left boundary) for the second condition which we can see in Fig. 2. For the first condition on the other hand the density never reaches the boundaries and is therefore preserved for the time of the simulation. When we review the Hamiltonian and our time evolution Eqs. 6 and 7, we see that the time evolution is not unitary. Thus, we are not guaranteed that the density is preserved. We still observe a linear progression for the second initial condition as about half of the density propagates to the right and accounts for some variance. Though, at each time step a certain fraction of the propagating density is send to the left which is eventually absorbed at the boundary and the further the density has propagated to the right the smaller density fractions are send to the left and the longer they have to travel until they reach the boundary. Thus, if we wouldn't constantly normalize the expectation value by the total density of the system we would see that at some point the variance will plateau to some value where the leakage rate becomes so low that we can't observe it anymore while the density still continues propagating to the right until eventually all density leaked out of the system.

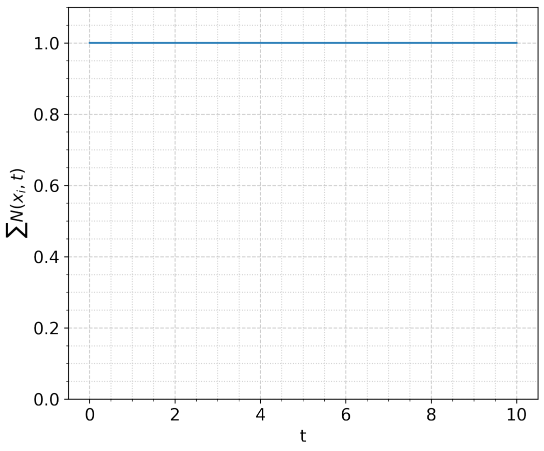


(a) Condition 1: Start in center $N(i_0, 0) = 1$ for $i_0 = (L+1)/2$.

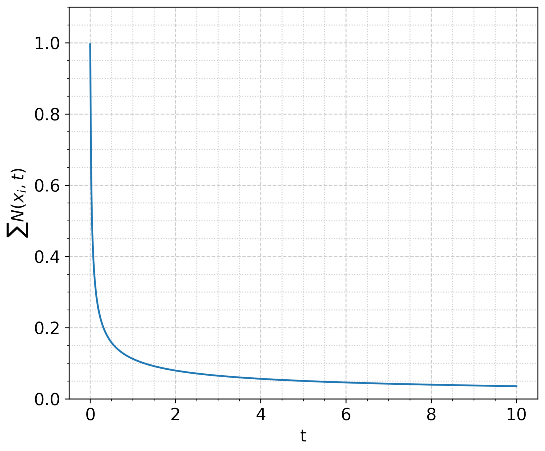


(b) Condition 2: Start at left $N(i_0, 0) = 1$ for $i_0 = 1$.

Figure 1. Product formula approach for two different initial conditions.



(a) Condition 1: Start in center $N(i_0, 0) = 1$ for $i_0 = (L+1)/2$.



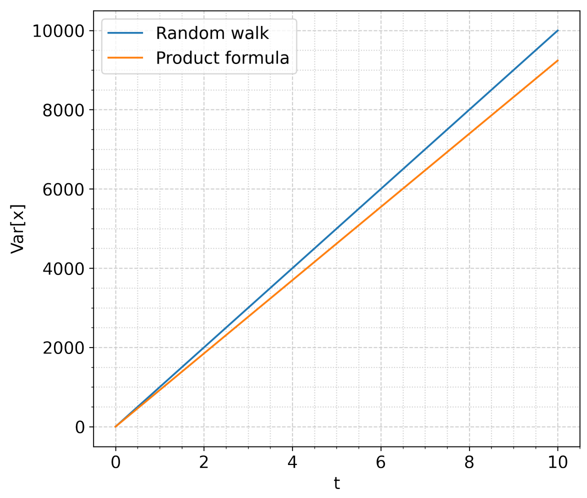
(b) Condition 2: Start at left $N(i_0, 0) = 1$ for $i_0 = 1$.

Figure 2. Time evolution of total density. For the first boundary conditions the particle density did not reach the boundaries after m time steps, and therefore the density is conserved over this time range. For the second boundary condition we see leakage over time at the left boundary.

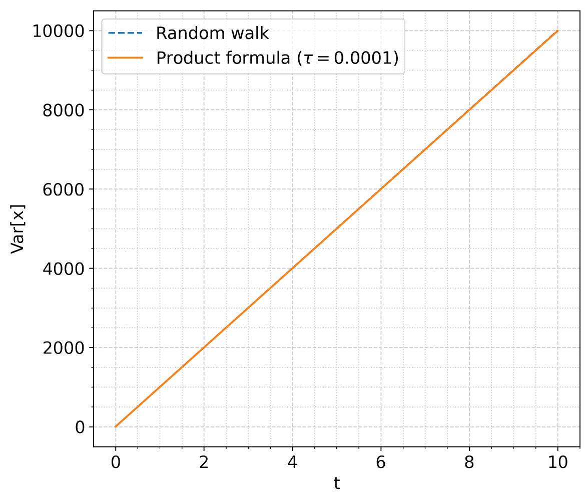
Finally, we compared the product formula simulation for $D = \Delta^2/2\tau = 5$ and otherwise identical settings as above for the first initial condition to the simulation results of 10000 random walkers. The results can be seen in Fig. 3a. As we can see both methods yield similar results. Although the error of the product formula approach increases with time, the general trend is the same. If we increase the time resolution from $\tau = 0.001$ to $\tau = 0.0001$ and consequently $m \to m = 100000$ we get a simulation result (Fig. 3b) which converges towards the Random Walk approach. Thus, in conclusion we can say that both simulation approaches, although very different in nature describe the phenomenon of diffusion equally well.

Discussion

In summary, we first derived in detail the update equations for the product formula approach for simulating the diffusion of particles. Afterwards, we showed which results one obtains for simulations starting with different sets of initial conditions.



(a) Time resolution of $\tau = 0.001$



(b) Increased time resolution of $\tau = 0.0001$

Figure 3. Comparison of product formula approach to Random Walk. For both simulations we used $T = 0, ..., 10$ with a time step $\tau = 0.001$ (3a) and $\tau = 0.0001$ (3b). We see increased convergence between the two methods if we increase the time resolution.

Afterwards, we showed that two seemingly very different approaches, namely the product formula and the Random Walk approach are both adequate descriptions of particle/mass diffusion. We showed that the product formula approach does not preserve particle density as the time-evolution is not unitary and that we therefore encounter leakage at the boundaries, which, for closed systems, is not desired. Furthermore, we showed that for increased time resolution the product formula approach converges towards the solution we obtained with the Random Walk approach. We omitted a detailed description of the random walk procedure as it was described in a previous report. Although our random walk algorithm differs slightly from our previous implementation it is formally equivalent.

Appendix

Common imports

```python
import numpy as np
import matplotlib.pyplot as plt

D = 1
L = 1001
delta = 0.1
tau = 0.001
m = 10_000

idx0 = L // 2 # init condition 1
# idx0 = 0 # init condition 2
```

Product formula approach

```python
from common import *

def expM(factor=1):
    '''Calculate exponential matrix e^aX. Also returns a.'''
    a = - tau * D / delta**2 * factor
    x = 1 + np.exp(2*a)
    y = 1 - np.exp(2*a)
    M = 0.5 * np.array([[x,y],[y,x]])
    return M, a

expB, aB = expM()
expA, aA = expM(factor=0.5)
```

```

N = np.zeros(L)
N[idx0] = 1 # start with all density at source location idx0

x1 = np.zeros(m) # Track 1st moment
x2 = np.zeros(m) # Track 2nd moment

diff0 = (np.array(range(L)) - idx0) * delta # Offset of each grid point to source

for i in range(m):

    ### Track moments ###

    x1[i] = np.sum(N * diff0) / np.sum(N) # 1st moment
    x2[i] = np.sum(N * diff0**2) / np.sum(N) # 2nd moment

    ### Update state ###

    # e^aA/2 @ N
    N[:-1] = np.hstack([eA @ v for v in np.split(N[:-1], 500)]).ravel()
    N[-1] *= np.exp(aA)

    # e^aB @ N
    N[1:] = np.hstack([eB @ v for v in np.split(N[1:], 500)]).ravel()
    N[0] *= np.exp(aB)

    # e^aA/2 @ N
    N[:-1] = np.hstack([eA @ v for v in np.split(N[:-1], 500)]).ravel()
    N[-1] *= np.exp(aA)

var = (x2 - x1**2) / delta**2 # Calculate variance
plt.plot(np.arange(0,m*tau, tau), var); # Plot variance over time range with tau steps
plt.xlabel('t');
plt.ylabel('Var[i(t)]');
```

Random Walk approach

```
from common import *

N = np.zeros(L) # concentration on grid position
N[idx0] = m # all particles in center

x1 = np.zeros(T) # store first moment
x2 = np.zeros(T) # store second moment

for i in range(m):
    x1[i] = np.sum(N / m * (np.array(range(L))-idx0)) # 1st moment
    x2[i] = np.sum(N / m * (np.array(range(L))-idx0)**2) # 2nd moment
    N[1:-1] = 0.5 * (N[:-2] + N[2:]) # update, fixed boundaries

var = x2 - x1**2 # calc variance
plt.plot(np.arange(0,m*tau,tau), var);
plt.plot(np.arange(0,m*tau,tau), '--');
plt.xlabel('t');
plt.ylabel('Var[x(t)]');
```