Diffusion of neurotransmitters in the synaptic cleft

FYS3150 - Project 5

Vilde Flugsrud - Candidate number 7
Mari Dahl Eggen - Candidate number 5

December 11, 2015

Contents

Abstract

In this project we are analyzing the quality of five numerical procedures for solving partial differential equations. The equation of interest is the diffusion equation, and the numerical methods we are testing are Forward Euler, Backward Euler, Crank-Nicolson, and two different Monte Carlo simulation approaches. We will see that the Crank-Nicolson scheme gives the best results, while the Monte Carlo simulations of partial differential equations in one dimension gives a relative error that is $\sim 10^3$ times bigger.

1 Introduction

Neurotransmitters are information carrying molecules that are used in transport of signals between neurons in the brain. They are transported from a presynaptic cell to a postsynaptic cell, across a synaptic cleft, which is the small space that separates the cell membranes of the two cells. In Figure 1 the cells and synaptic cleft are illustrated. In this project we are interested in how the neurotransmitters are moving across the synaptic cleft. We assume that they move due to diffusion, and will therefore solve the diffusion equation to figure out how the system evolves with time. The diffusion equation is a partial differential equation, and it exists many numerical methods for solving it. The equation is in this case also analytically solvable, which makes it easy to analyze the quality of the numerical methods. In this project we will test five different numerical methods, and compare the results from each, to determine which of the methods is the best.

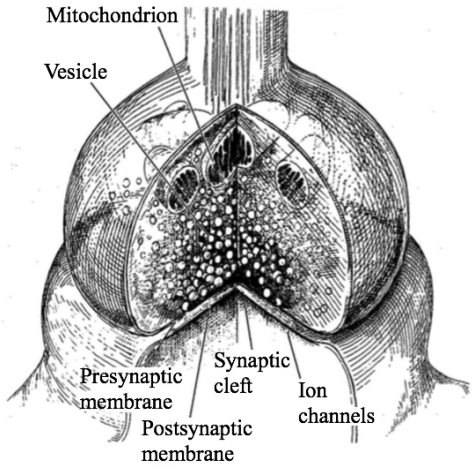


Figure 1: Illustration of how the presynaptic and postsynaptic cell are lying against each other, when the presynaptic cell are sending signals to the postsynaptic cell through the synaptic cleft. The synaptic cleft is the small space between the cell membranes. (From task text for project 5 (edited).)

2 Theory

2.1 Analytical solution of the diffusion equation

We will now find a closed-form solution of the diffusion equation in one dimension, where the diffusion constant is set to $D = 1$, so the partial differential equation to solve is

$$\frac{\partial^2 u}{\partial x^2} = \frac{\partial u}{\partial t}. \tag{1}$$

The initial condition is

$$u(x, 0) = 0 \quad \text{for} \quad 0 < x < 1,$$

and the boundary conditions are

$$u(0, t) = 1 \quad \text{for} \quad t > 0, \quad \text{and} \quad u(1, t) = 0 \quad \text{for} \quad t > 0.$$

We are given the steady-state solution of the Eq. (1) for our given boundary conditions, that is, the solution of the system when u is independent of time. It is $u_s(x) = 1 - x$. To solve the time dependent equation we introduce a new function $v(x, t) = u(x, t) - u_s(x)$, so that we get the new initial condition

$$v(x, 0) = -u_s(x) \quad \text{for} \quad 0 < x < 1, \tag{2}$$

and the new boundary conditions

$$v(0, t) = 0 \quad \text{for} \quad t > 0, \quad \text{and} \quad v(1, t) = 0 \quad \text{for} \quad t > 0. \tag{3}$$

We can use the function $v(x, t)$ because

$$\frac{\partial^2 (u(x, t) - u_s(x))}{\partial x^2} = \frac{\partial (u(x, t) - u_s(x))}{\partial t} \quad \Rightarrow \quad \frac{\partial^2 u}{\partial x^2} = \frac{\partial u}{\partial t}.$$

We see that the general solution we find for $u(x,t)$ also will hold for $v(x,t)$. To find a solution we make the ansatz

$$u(x,t) = z(x)w(t),$$

so

$$\frac{\partial^2 u}{\partial x^2} = \frac{\partial u}{\partial t} \quad \Rightarrow \quad u_{xx} = u_t \quad \Rightarrow \quad wz_{xx} = zw_t \quad \Rightarrow \quad \frac{z_{xx}}{z} = \frac{w_t}{w}.$$

Now the left hand side and the right hand side of the equal sign is independent of each other, so they have to be constant. We introduce the constant $-\lambda^2$, so that

$$\frac{z_{xx}}{z} = \frac{w_t}{w} = -\lambda^2,$$

and then the two differential equations

$$z_{xx} + \lambda^2 z = 0 \quad \text{and} \quad w_t + \lambda^2 w = 0$$

are left to be solved. These are two standard differential equations with general solutions

$$z(x) = A\cos(\lambda x) + B\sin(\lambda x) \quad \text{and} \quad w(t) = Ce^{-\lambda^2 t}.$$

If we put these solutions together we find the general solution of our problem in Eq. (1), for the new function $v(x,t)$.

$$v(x,t) = u(x,t) - u_s(x) = \left(A\cos(\lambda x) + B\sin(\lambda x)\right) Ce^{-\lambda^2 t} \tag{4}$$

Now we can use the boundary conditions in Eq. (3) to find the closed-form solution of $v(x,t)$.

$$v(0,t) = \left(A\cos(0) + B\sin(0)\right) Ce^{-\lambda^2 t} = ACe^{-\lambda^2 t} = 0 \quad \Rightarrow \quad A = 0$$

$$v(1,t) = \left(A\cos(\lambda) + B\sin(\lambda)\right) Ce^{-\lambda^2 t} = B\sin(\lambda)Ce^{-\lambda^2 t} = 0$$

$$\Rightarrow \quad \sin(\lambda) = 0 \quad \Rightarrow \quad \lambda_n = n\pi \quad \text{for} \quad n = 1, 2, 3, ...$$

Then we have

$$v_n(x,t) = B_n C_n \sin(n\pi x) e^{-(n\pi)^2 t} = A_n \sin(n\pi x) e^{-(n\pi)^2 t},$$

and

$$v(x,t) = \sum_{n=1}^{\infty} A_n \sin(n\pi x) e^{-(n\pi)^2 t}$$

At last we can use the initial condition in Eq. (2) to find an expression for the n-dependent constant A_n.

$$v(x,0) = \sum_{n=1}^{\infty} A_n \sin(n\pi x) = -u_s(x) = x - 1$$

We recognize this as a Fourier series. In general we have

$$f(x) = \frac{a_0}{2} + \sum_{n=1}^{\infty} \left(a_n \cos\left(\frac{n\pi x}{P}\right) + b_n \sin\left(\frac{n\pi x}{P}\right)\right) = \sum_{n=1}^{\infty} b_n \sin\left(n\pi x\right), \tag{5}$$

where we have defined $a_0 = a_n = 0$ and $P = 1$. In the Fourier series b_n is defied as

$$b_n = \frac{2}{P} \int_{x_0}^{x_0+P} f(x) \cdot \sin\left(\frac{n\pi x}{P}\right) dx. \tag{6}$$

If we use Eq. (5) and (6) in our problem to find A_n we get

$$A_n = 2 \int_0^1 (x-1) \sin\left(n\pi x\right) dx = 2 \left[\int_0^1 x \sin(n\pi x) dx - \int_0^1 \sin(n\pi x) dx \right],$$

and by use of integration by parts we get

$$= 2\left[\left[-\frac{x}{n\pi}\cos(n\pi x)\right]_0^1 + \frac{1}{n\pi}\int_0^1 \cos(n\pi x)dx - \frac{1}{n\pi}\left[-\cos(n\pi x)\right]_0^1\right]$$

$$= 2\left[\left(-\frac{1}{n\pi}\cos(n\pi)\right) + \left[\frac{1}{n\pi}\cdot\frac{1}{n\pi}\sin(n\pi x)\right]_0^1 + \frac{1}{n\pi}\left(\cos(n\pi) - \cos(0)\right)\right]$$

$$= 2\left[-\frac{(-1)^n}{n\pi} + \frac{1}{n^2\pi^2}\left(\sin(n\pi) - \sin(0)\right) + \frac{1}{n\pi}\left((-1)^n - 1\right)\right]$$

$$= 2\left[-\frac{(-1)^n}{n\pi} + \frac{(-1)^n}{n\pi} - \frac{1}{n\pi}\right] = -\frac{2}{n\pi},$$

which gives us the closed-form solution

$$v(x,t) = -\frac{2}{\pi}\sum_{n=1}^{\infty}\frac{1}{n}\sin\left(n\pi x\right)e^{-(n\pi)^2 t}. \tag{7}$$

This gives us the concentration of the neurotransmitters in the synaptic cleft as

$$u(x,t) = 1 - x - \frac{2}{\pi}\sum_{n=1}^{\infty}\frac{1}{n}\sin\left(n\pi x\right)e^{-(n\pi)^2 t}.$$

2.2 Numerical solution of the diffusion equation

2.2.1 The explicit Forward Euler algorithm

In the explicit forward Euler algorithm we have that

$$\frac{\partial u}{\partial t} = u_t \approx \frac{u(x_i, t_j + \Delta t) - u(x_i, t_j)}{\Delta t} = \frac{u_i^{j+1} - u_i^j}{\Delta t} \tag{8}$$

$$\frac{\partial^2 u}{\partial x^2} = u_{xx} \approx \frac{u(x_i + \Delta x, t_j) - 2u(x_i, t_j) + u(x_i - \Delta x, t_j)}{\Delta x^2} \tag{9}$$

$$= \frac{u_{i+1}^j - 2u_i^j + u_{i-1}^j}{\Delta x^2},$$

which gives us Eq. (1) on the form

$$\frac{u_i^{j+1} - u_i^j}{\Delta t} = \frac{u_{i+1}^j - 2u_i^j + u_{i-1}^j}{\Delta x^2}$$

$$\Rightarrow \quad u_i^{j+1} = u_i^j + \alpha \left(u_{i+1}^j - 2u_i^j + u_{i-1}^j \right) \tag{10}$$

where $\alpha = \frac{\Delta t}{\Delta x^2}$. This is an explicit formula for the unknown, which is the value of the concentration at the time $j+1$, and it can thus be calculated directly.

2.2.2 The implicit Backward Euler algorithm

The partial derivatives are in the implicit backward Euler algorithm given by

$$u_t \approx \frac{u(x_i, t_j) - u(x_i, t_j - \Delta t)}{\Delta t} = \frac{u_i^j - u_i^{j-1}}{\Delta t} \tag{11}$$

$$u_{xx} \approx \frac{u(x_i + \Delta x, t_j) - 2u(x_i, t_j) + u(x_i - \Delta x, t_j)}{\Delta x^2} \tag{12}$$

$$= \frac{u_{i+1}^j - 2u_i^j + u_{i-1}^j}{\Delta x^2},$$

which gives us Eq. (1) on the form

$$\frac{u_i^j - u_i^{j-1}}{\Delta t} = \frac{u_{i+1}^j - 2u_i^j + u_{i-1}^j}{\Delta x^2}$$

$$\Rightarrow \quad u_i^{j-1} = u_i^j - \alpha \left(u_{i+1}^j - 2u_i^j + u_{i-1}^j \right), \tag{13}$$

where $\alpha = \frac{\Delta t}{\Delta x^2}$. This equation can be represented as a matrix equation on the form

$$U_{j-1} = \hat{A}U_j = (\mathbb{1} + \alpha\hat{B})U_j, \tag{14}$$

where

$$U_j = \begin{bmatrix} u_1^j \\ u_2^j \\ \vdots \\ u_n^j \end{bmatrix}, \quad U_{j-1} = \begin{bmatrix} u_1^{j-1} \\ u_2^{j-1} \\ \vdots \\ u_n^{j-1} \end{bmatrix}, \quad \hat{B} = \begin{bmatrix} 2 & -1 & 0 & 0 & \dots & 0 \\ -1 & 2 & -1 & 0 & & \vdots \\ 0 & -1 & \ddots & \ddots & & \vdots \\ \vdots & & \ddots & \ddots & \ddots & \vdots \\ \vdots & & & \ddots & \ddots & -1 \\ 0 & \dots & \dots & \dots & -1 & 2 \end{bmatrix}, \tag{15}$$

and $\mathbb{1}$ is the identity matrix. Our unknown points are stored in U_j, and we can find them my solving the matrix equation

$$U_j = (\mathbb{1} + \alpha\hat{B})^{-1}U_{j-1}. \tag{16}$$

2.2.3 The implicit Crank-Nicolson scheme

In the Crank-Nicolson scheme, the derivatives we are looking at in this project, are given by

$$u_t \approx \frac{u(x_i, t_j + \Delta t) - u(x_i, t_j)}{\Delta t} = \frac{u_i^{j+1} - u_i^j}{\Delta t} \tag{17}$$

$$u_{xx} \approx \frac{1}{2}\left(\frac{u(x_i + \Delta x, t_j) - 2u(x_i, t_j) + u(x_i - \Delta x, t_j)}{\Delta x^2} + \right. \tag{18}$$

$$\left. \frac{u(x_i + \Delta x, t_j + \Delta t) - 2u(x_i, t_j + \Delta t) + u(x_i - \Delta x, t_j + \Delta t)}{\Delta x^2}\right).$$

$$= \frac{1}{2}\left(\frac{u_{i+1}^j - 2u_i^j + u_{i-1}^j}{\Delta x^2} + \frac{u_{i+1}^{j+1} - 2u_i^{j+1} + u_{i-1}^{j+1}}{\Delta x^2}\right),$$

which gives us Eq. (1) on the form

$$\frac{u_i^{j+1} - u_i^j}{\Delta t} = \frac{1}{2}\left(\frac{u_{i+1}^j - 2u_i^j + u_{i-1}^j}{\Delta x^2} + \frac{u_{i+1}^{j+1} - 2u_i^{j+1} + u_{i-1}^{j+1}}{\Delta x^2}\right)$$

$$\Rightarrow \quad 2u_i^{j+1} - 2u_i^j = \alpha\left(u_{i+1}^j - 2u_i^j + u_{i-1}^j + u_{i+1}^{j+1} - 2u_i^{j+1} + u_{i-1}^{j+1}\right)$$

$$\Rightarrow \quad -\alpha u_{i-1}^j + (-2+2\alpha)u_i^j - \alpha u_{i+1}^j = \alpha u_{i-1}^{j+1} + (-2-2\alpha)u_i^{j+1} + \alpha u_{i+1}^{j+1}$$

$$\Rightarrow \quad (-2\mathbb{1} + \alpha\hat{B})U_j = (-2\mathbb{1} - \alpha\hat{B})V_{j+1} \quad \Rightarrow \quad (2\mathbb{1} - \alpha\hat{B})U_{j-1} = (2\mathbb{1} + \alpha\hat{B})U_j$$

$$U_j = (2\mathbb{1} + \alpha\hat{B})^{-1}(2\mathbb{1} - \alpha\hat{B})U_{j-1}, \tag{19}$$

where U_j, U_{j-1} and $\hat{B}$ is given in Eq. (15) and $\mathbb{1}$ is the identity matrix. We need to solve this matrix equation to find our unknown points.

2.2.4 Monte Carlo Theory

The second approach to our problem is a Monte Carlo simulation with random walks. This is a method closely related to diffusion for the following reasons. The framework of random walks allows for a microscopic description of Brownian motion. Brownian motion is the behavior exhibited by small fractions of any system when exposed to random fluctuations of the medium - generally it is the random movement of particles. Diffusion can be the behavior of a large number of particles subjected to Brownian motion. While the behavior may be truly random at the microscopic level of each particle, the behavior at the macroscopic level of the system becomes seemingly deterministic. This is because macroscopic system states which are only obtained from certain combinations of particle movements become infinitesimally unlikely compared to system states that are possible results of various different motions of the particles. This is equivalent to the fact that entropy increases. Diffusion results from the overwhelming probability of movement towards lower concentration.

In our case the diffusing particles are the neurotransmitters navigating the synaptic cleft. This leads to the idea of achieving a simulation of the diffusion process by modeling the neurotransmitters with random walkers. A Markov process is a simple random walk in that two conditions for the microscopic movements are met - ergodicity and detailed balance, assuming there is an equal probability for every microscopic motion and that the probabiltity is also equal to that of the reverse motion, respectively. Underlining the close

link between Markov chains and diffusion is the fact that the discretized diffusion equation can be derived from a Markov process. Or stated differently, a Markov process yields in the limit of infinitely many steps of the diffusion equation. [2]
It can be shown that for the diffusion process the root mean square displacement after a time Δt is

$$\sqrt{\langle x^2 \rangle - \langle x \rangle^2} = \sqrt{2D\Delta t}. \tag{20}$$

We will implement this as our step length l_0 when developing the algorithm.

2.3 Truncation errors

The basic idea behind Forward Euler, Backward Euler and Crank-Nicolson scheme is to advance a solution at point $u(x_i, t_j)$, to a solution at point $u(x_{i+1}, t_{j+1})$. This three methods stems from the linear Taylor polynomial

$$f(x) = \sum_{n=0}^{\infty} \frac{f^{(n)}(a)}{n!}(x-a)^n, \tag{21}$$

and therefore we can use it to estimate the truncation error in the three methods. Since we are looking at a function of one dimension in space and one dimension in time, we have to find the truncation error for both separately. The resulting truncation errors are listed in Table 1.

2.3.1 Forward Euler

First we will find the local truncation error in time. Then we need to find a expression for Eq. (8), by use of Eq. (21). We are looking for a solution at point u_i^{j+1}, so that is the point we will express as a Taylor polynomial.

$$u_i^{j+1} = u_i^j + (u_i^j)_t \Delta t + \frac{(u_i^j)_{tt}}{2}\Delta t^2 + \dots$$

$$\Rightarrow \quad (u_i^j)_t \simeq \frac{u_i^{j+1} - u_i^j}{\Delta t} - \frac{(u_i^j)_{tt}}{2}\Delta t = \frac{u_i^{j+1} - u_i^j}{\Delta t} - \mathcal{O}(\Delta t)$$

We see that the biggest error term in Eq. (8) is of the order $\mathcal{O}(\Delta t)$. Now we will do the same procedure, a bit extended, to find the local truncation error in space.

$$I: \quad u_{i+1}^j = u_i^j + (u_i^j)_x \Delta x + \frac{(u_i^j)_{xx}}{2}\Delta x^2 + \frac{(u_i^j)_{xxx}}{6}\Delta x^3 + \frac{(u_i^j)_{xxxx}}{24}\Delta x^4 + \dots$$

$$II: \quad u_{i-1}^j = u_i^j - (u_i^j)_x \Delta x + \frac{(u_i^j)_{xx}}{2}\Delta x^2 - \frac{(u_i^j)_{xxx}}{6}\Delta x^3 + \frac{(u_i^j)_{xxxx}}{24}\Delta x^4 + \dots$$

$$I + II: \quad u_{i+1}^j + u_{i-1}^j \simeq 2u_i^j + (u_i^j)_{xx}\Delta x^2 + \frac{(u_i^j)_{xxxx}}{12}\Delta x^4$$

$$(u_i^j)_{xx} \simeq \frac{u_{i+1}^j - 2u_i^j + u_{i-1}^j}{\Delta x^2} - \frac{(u_i^j)_{xxxx}}{12}\Delta x^2 = \frac{u_{i+1}^j - 2u_i^j + u_{i-1}^j}{\Delta x^2} - \mathcal{O}(\Delta x^2)$$

The biggest error term in Eq. (9) is then of order $\mathcal{O}(\Delta x^2)$.

2.3.2 Backward Euler

We use the same procedure as we did in the previous section to find the local truncation error in time for Backward Euler. We then want to find an expression for Eq. (11).

$$u_i^{j-1} = u_i^j - (u_i^j)_t \Delta t + \frac{(u_i^j)_{tt}}{2}\Delta t^2 + \dots$$

$$\Rightarrow \quad (u_i^j)_t = \frac{u_i^j - u_i^{j-1}}{\Delta t} + \frac{(u_i^j)_{tt}}{2}\Delta t = \frac{u_i^j - u_i^{j-1}}{\Delta t} + \mathcal{O}(\Delta t)$$

When it comes to the local truncation error in space we see that Eq. (12) is the same as Eq. (9), and so the local truncation error is also the same. See the derivation in the previous section, where we find that the biggest error term in space is of order $\mathcal{O}(\Delta x^2)$.

2.3.3 Crank-Nicolson scheme

The derivation of the local truncation error in space and time for the Crank-Nicolson scheme is equivalent to the derivations we did in the two previous sections, but with some more calculations, that we pass in this project. The derivation can be seen in REF... It appears that the biggest error term in Eq. (17) is $\mathcal{O}(\Delta t^2)$, and $\mathcal{O}(\Delta x^2)$ in Eq. (18).

2.4 Stability properties

We can investigate the stability properties of Forward Euler, Backward Euler and Crank-Nicolson scheme by use of the spectral radius of the coefficient matrix for each of the algorithms. The spectral radius is given by

$$\rho(\hat{A}) = \{|\lambda| : \quad \det(\hat{A} - \lambda \mathbb{1})\},$$

which means that the spectral radius is equal to the maximum eigenvalue of the given matrix $\hat{A}$. If the restriction

$$\rho(\hat{A}) < 1 \tag{22}$$

is fulfilled, we know that the system we are looking at can reach a steady state, which must be the case in this project. The restrictions found for the algorithms in this section are listed in Table 1.

2.4.1 Implicit Backward Euler

In 2.2.2 we showed that Eq. (1) can be solved by use of the Backward Euler method, and that the algorithm has the form as we can see in Eq. (16). We then have the coefficient matrix $\hat{A}^{-1} = (\mathbb{1} + \alpha\hat{B})^{-1}$, and want to find its eigenvalues $\lambda_{\hat{A}^{-1}}$, to find the maximum eigenvalue, so we can investigate the algorithms stability properties. Since we have the relation

$$\hat{A}v_k = \lambda_k v_k \quad \Rightarrow \quad \hat{A}^{-1}\hat{A}v_k = \lambda_k \hat{A}^{-1}v_k$$

$$\Rightarrow \quad \mathbb{1}v_k = \lambda_k \hat{A}^{-1}v_k \quad \Rightarrow \quad \frac{1}{\lambda_k} = \hat{A}^{-1}v_k,$$

we have that $\lambda_{\hat{A}^{-1}} = \frac{1}{\lambda_{\hat{A}}}$, so the easiest is to find $\lambda_{\hat{A}}$ first. We have that

$$\lambda_{\hat{A}} = \lambda_{\mathbb{1}} + \alpha\lambda_{\hat{B}}, \tag{23}$$

where $\lambda_{\mathbb{1}} = 1$ are the eigenvalues of the identity matrix and $\lambda_{\hat{B}}$ are the eigenvalues of $\hat{B}$, a matrix which can be seen in Eq. (15). $\hat{B}$ is tridiagonal and Toeplitz, which means that the eigenvalues has a closed form solution [5], [6]

$$\lambda_k = a + 2\sqrt{bc}\cos\left(\frac{k\pi}{n+1}\right), \quad \text{for} \quad k = 1, \ldots, n,$$

where a are the elements on the diagonal, b and c are the elements in the diagonal above and below the mid diagonal respectively, and n is the size of the square matrix $\hat{B}$. The eigenvalues of $\hat{B}$ are then given by

$$\lambda_k = 2 + 2\sqrt{(-1)(-1)}\cos\left(\frac{k\pi}{n+1}\right) = 2 + 2\cos(\theta_k) = 2(1 + \cos(\theta_k)), \tag{24}$$

where we have defined $\theta_k = \frac{k\pi}{n+1}$. Now we can find the eigenvalues of $\hat{A}$ by use of Eq. (26), and thus find the eigenvalues of $\hat{A}^{-1}$.

$$\lambda_{\hat{A},k} = 1 + 2\alpha(1 + \cos(\theta_k))$$

We know that $-1 < \cos(\theta_k) < 1$ and $\alpha > 0$, which leads to that we always have $\lambda_{\hat{A},k} > 1$. This leads to that all the eigenvalues of $\hat{A}^{-1}$ are

$$\lambda_{\hat{A}^{-1},k} = \frac{1}{\lambda_{\hat{A},k}} = \frac{1}{1 + 2\alpha(1 + \cos(\theta_k))} < 1,$$

which means that the biggest eigenvalue of $\hat{A}^{-1}$ is less that one. Because of this Eq. (22) is always fulfilled, and we have no restrictions on α for this algorithm.

2.4.2 Explicit Forward Euler

By looking at Eq. (10) it is easy to realize that it can be rewritten to a matrix equation on the same form as Eq. (13), by just a few changes. The matrix equation is

$$U_j = \hat{D}U_{j-1} = (\mathbb{1} - \alpha\hat{B})U_{j-1},$$

where U_j, U_{j-1} and $\hat{B}$ are given in Eq. (15), and $\mathbb{1}$ is the identity matrix. Now we want to find the eigenvalues of $\hat{D}$, so that we can find its biggest eigenvalue, and then see if there is any restrictions to follow when the algorithm is executed. The eigenvalues of $\hat{D}$ is given by

$$\lambda_{\hat{D}} = \lambda_{\mathbb{1}} - \alpha\lambda_{\hat{B}},$$

where $\lambda_{\mathbb{1}} = 1$ are the eigenvalues of the identity matrix and $\lambda_{\hat{B}}$ are the eigenvalues of $\hat{B}$. The eigenvalues of $\hat{B}$ are given in Eq. (24), and by use of this we find the eigenvalues of $\hat{D}$ to be

$$\lambda_{\hat{D},k} = 1 - 2\alpha(1 + \cos(\theta_k))$$

We can see that the smallest possible value of θ_k gives us the biggest eigenvalue of $\hat{D}$. Since $\theta_k = \frac{k\pi}{n+1}$ for $k = 1, \ldots, n$, θ_k can not be zero, but we suppose that n is big so that the minimum value is $\theta_k \simeq 0$. If we insert this into the previous equation and uses the restriction in Eq. (22), we can find the restriction on α, so that we are sure to reach a steady state by use of the Backward Euler method.

$$-1 < (1 - 2\alpha(1 + \cos(0))) < 1 \quad \Rightarrow \quad -1 < (1 - 4\alpha) < 1$$

$$(I): \quad -1 < 1 - 4\alpha \quad \Rightarrow \quad \alpha < \frac{1}{2}$$

$$(II): \quad 1 - 4\alpha < 1 \quad \Rightarrow \quad \alpha > 0$$

Restriction (II) is already fulfilled, because $\alpha = \frac{\Delta t}{\Delta x^2}$, and the step lengths will always be defined as positive. Restriction (I) will however put a restriction on the magnitude of the step lengths of time and space. We get that

$$\frac{\Delta t}{\Delta x^2} < \frac{1}{2} \quad \Rightarrow \quad \Delta t < \frac{\Delta x^2}{2}. \tag{25}$$

2.4.3 Implicit Crank-Nicolson scheme

In 2.2.3 we showed that Eq. (1) can be solved by use of the Crank-Nicolson scheme, and that the algorithm has the form as we can see in Eq. (19). We then have the coefficient matrix $\hat{C} = (2\mathbb{1} + \alpha\hat{B})^{-1}(2\mathbb{1} - \alpha\hat{B})$, and want to find its eigenvalues $\lambda_{\hat{C}}$, to find the maximum eigenvalue, so we can investigate the algorithms stability properties. The eigenvalues are given by

$$\lambda_{\hat{C}} = (2\lambda_{\mathbb{1}} + \alpha\lambda_{\hat{B}})^{-1}(2\lambda_{\mathbb{1}} - \alpha\lambda_{\hat{B}}) = \lambda_{C_1}\lambda_{C_2}, \tag{26}$$

where it is easy to realize that

$$\lambda_{\hat{C_1}} = \frac{1}{2 + 2\alpha(1 + \cos(\theta_k))} \quad \text{and} \quad \lambda_{\hat{C_2}} = 2 - 2\alpha(1 + \cos(\theta_k))$$

if we take a look at 2.4.1 and 2.4.2 respectively. We then have the eigenvalues

$$\lambda_{\hat{C}} = \frac{2 - 2\alpha(1 + \cos(\theta_k))}{2 + 2\alpha(1 + \cos(\theta_k))} = \frac{2 - \mu_k}{2 + \mu_k}.$$

To fulfill the restriction in Eq. (22), the maximum eigenvalue have to fulfill

$$-1 < \frac{2 - \mu_k}{2 + \mu_k} < 1$$

$$(I): \quad -1 < \frac{2 - \mu_k}{2 + \mu_k} \quad \Rightarrow \quad -2 - \mu_k < 2 - \mu_k \quad \Rightarrow \quad -2 < 2$$

$$(II): \quad \frac{2 - \mu_k}{2 + \mu_k} < 1 \quad \Rightarrow \quad 2 - \mu_k < 2 + \mu_k \quad \Rightarrow \quad \mu_k > 0$$

We can see that (I) always is fulfilled, but we have to take a closer look at (II). $\mu_k = 2\alpha(1 + \cos(\theta_k))$ and $\theta_k = \frac{k\pi}{n+1}$ for $k = 1, \ldots, n$. If we suppose that n is big, we see that the smallest possible value of θ_k is $\theta_k \simeq 0$ and that its biggest possible value is $\theta_k \simeq \pi$. We insert these boundaries into (II), to find the restrictions of the Clark-Nicolson scheme.

$$(i): 2\alpha(1 + \cos(0)) > 0 \quad \Rightarrow \quad 4\alpha > 0$$

$$(ii): 2\alpha(1 + \cos(\pi)) > 0 \quad \Rightarrow \quad 4\alpha > 0 \quad \Rightarrow \quad 2\alpha(1 - 1) > 0$$

We always have that $\alpha = \frac{\Delta t}{\Delta x^2} > 0$, because the step lengths always are defined to be positive. Then it is easy to see that (i) always is fulfilled. The expression in (ii) indicates that $0 > 0$, but this is never going to be the case since θ_k never is going to reach the value π exactly, it will just approach π, so we will always have that $\cos(\theta_k) < 1$. This means that (ii) always is fulfilled as well, and we can conclude that the Crank-Nicolson scheme do not have any restrictions, which means that the system always will reach a steady state.

3 Method

3.1 Simulation of diffusion of neurotransmitters

The transport process of the neurotransmitters in the synaptic cleft is governed by diffusion, and can therefore be described mathematically by the diffusion equation

$$D\nabla^2 u = \frac{\partial u}{\partial t}$$

where u is the concentration of particular neurotransmitters, and D is the diffusion constant in this particular synaptic cleft. To simplify the problem we assume that the presynaptic cell releases the neurotransmitters roughly at the same time along its membrane, and that the whole synaptic cleft has approximately the same width everywhere. Then, because the area of the synaptic cleft is so large compared to its width, we can assume that the neurotransmitter concentration only varies in the direction from the presynaptic cell to the postsynaptic cell. This gives us the opportunity to look at the problem in only one spatial direction. We choose the direction from the presynaptic cell to the postsynaptic cell to be the x-direction, and the diffusion equation reduces to

$$D\frac{\partial^2 u}{\partial x^2} = \frac{\partial u}{\partial t}. \tag{27}$$

The simplified system is shown in Figure 2, and we see that the neurotransmitters are released from the presynaptic cell at $x = 0$, and absorbed by the postsynaptic cell at $x = d$.

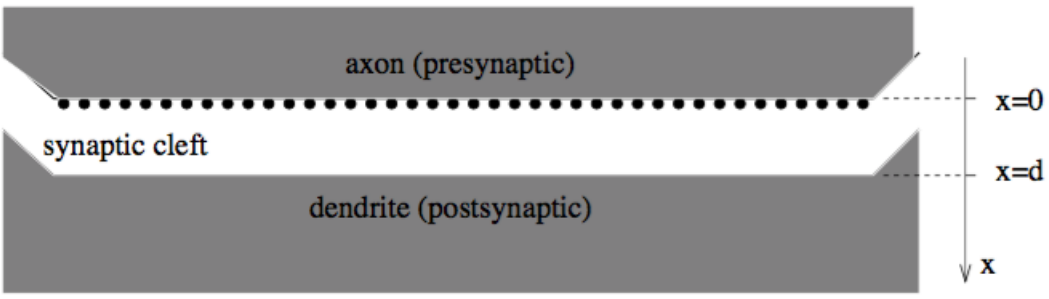


Figure 2: A model of the simplified system of two cells that are interchanging neurotransmitters. The neurotransmitters are sent from the presynaptic cell at $x = 0$, through the synaptic cleft, to the postsynaptic cell at $x = d$. (From assignment text for project 5.)

3.2 Implementation of algorithms

The diffusion equation (27) can be solved in several ways, and in this project we are solving it with five different methods. In this section the implementation of the five methods is explained. In the implementation of the following three methods we used the substitution $v(x,t) = u(x,t) - u_s(x,t)$ outlined in the theory section. This simplifies the initial condition to Eq. (2) and the boundary conditions to Eq. (3).

3.2.1 Forward Euler

This algorithm has a simple implementation as v_i^{j+1} is explicitly given by Eq. (10). We compute the new distribution at every time step j with the code

```
for (int i=1; i<Nx; i++) {
    v(i) = alpha*v_prev(i-1) + (1-2*alpha)*v_prev(i) + alpha*v_prev(i
        +1);
}
```

3.2.2 Backward Euler

As described in the theory section we find the solution by solving the matrix Eq. (14) at every time step j. As seen the solution is given implicitly, but rather than inverting the matrix we solve it on the form $(\mathbb{1} + \alpha\hat{B})V_j = V_{j-1}$ with Gaussian elimination using row operations. The simple form of the three diagonal matrix means that the row operations are fairly easily implemented. The lower non-diagonals $a = -\alpha$ turn out zero while the upper non-diagonals $c = -\alpha$ are unchanged. The diagonals $b = 1 + 2\alpha$ and the right hand side v_i^{j-1} are changed accordingly:

$$b_{i_{new}} = b - \frac{a \cdot c}{b_{i-1_{new}}} \quad \text{and} \quad v_{i_{new}}^{j-1} = v_i^{j-1} - \frac{a \cdot v_{i-1_{new}}^{j-1}}{b_{i-1_{new}}},$$

where *new* signifies the value after a row operation. This constitutes the forward substitution. These lines are implemented in the code as

```
// Forward substitution
v_prev_rowoperated(1) = v_prev(1);
b_rowoperated(1) = b;
for (int i=2; i<Nx; i++) {
    b_rowoperated(i) = b - ac/b_rowoperated(i-1);
    v_prev_rowoperated(i) = v_prev(i) - a_c*v_prev_rowoperated(i-1)/
        b_rowoperated(i-1);
}
```

The backward substitution to finally obtain v_i^j is completed by computing

$$v_i^j = \frac{v_{i_{new}}^{j-1} - c \cdot v_{i+1}^j}{b_{i_{new}}},$$

which is implemented in the code as

```
// Backward substitution
v(Nx-1) = v_prev_rowoperated(Nx-1)/b_rowoperated(Nx-1);
for (int i=Nx-2; i>0; i--) {
    v(i) = (v_prev_rowoperated(i)-a_c*v(i+1))/b_rowoperated(i);
}
```

These computations are done in a loop iterating over every time step j to obtain the final vector $V_{N_t} = v(x, t = T)$.

3.2.3 Crank-Nicolson

From Eq. (19) we have that

$$(2\mathbb{1} + \alpha\hat{B})V_j = (2\mathbb{1} - \alpha\hat{B})V_{j-1}.$$

Thus V_j can be obtained by first computing the vector on the right hand side and solving the resulting linear equations problem. This is explicitly given and so the method from the Forward Euler procedure can be applied here. After this we are left with an implicit problem similar to the Backward Euler method and we solve it by the same procedure of Gaussian elimination. At each time step j we then have a two step procedure combining the code from Forward and Backward Euler.

3.2.4 Monte Carlo simulation with discrete positions

In the first implementation of the Monte Carlo scheme the set of possible particle positions is discretized. The particles move with a constant step length $l_0 = \sqrt{2D\Delta t}$ with equal probability for moving left and right.

The algorithm is implemented as two for loops. The outer loop iterates over the given number of Monte Carlo cycles, each cycle representing a new particle. The inner loop iterates over the given number of time steps. Thus it simulates a random walk for each particle, moving either left or right at every time step. If the particle moves outside the synaptic cleft the loop breaks. This accounts for the boundary conditions. While the result is an array representing the concentration of neurotransmitters at a final time T, we count the transmitter's position at every time step. This is meant as a

computationally efficient way of accounting for the constant stream of new transmitters released into the cleft. The transmitter's movement at a time earlier than T might well represent the movement of a transmitter that entered the cleft at a later time. The code for the random walk loop is as follows

```
for (double time = dt; time <= T; time+=dt) {
    if (ran0(&idum) <= move_prob) { // move to the right
        pos_index += 1;
        // if it has reached the end, stop counting it
        if (pos_index >= Nx) break;
    } else {                        // move to the left
        pos_index -= 1;
        // if it has returned to start, stop counting it
        if (pos_index <=0) break;
    }
    // count the particle's position
    pos_count(pos_index) += 1;
}
```

3.2.5 Monte Carlo simulation with continuous positions

This method implements the Monte Carlo scheme with a continuous set of positions. This is done by changing the step length to $l_0 = \xi\sqrt{2D\Delta t}$ where ξ is a random number chosen from a Gaussian distribution with mean value 0 and standard deviation 1. See the code for the implementation. The initial positions of the particles are given by a randomly distributed position within the first interval, as opposed to them initially being set to 0. There exists some alternative approaches here, such as initiating the particles at an interval left of zero, or in a small interval centered at zero. The initial condition is given as the Dirac delta function and it is suggested that the problem is how to represent this well at continuous values.

3.3 Unit tests and verification of results

To be sure that the results we get from our numerical calculations are correct, it is necessary to include some tests in the code. If there is some of the pre calculations in the algorithm that we know the answer of, or if there is some restrictions on the pre calculations, these are the parts we want to check with tests.

3.3.1 Unit tests in Forward Euler, Backward Euler and the Crank-Nicolson scheme

In 2.4.1, 2.4.2 and 2.4.3 we were looking for restrictions on the step lengths in time and space, from the restriction in Eq. (22). The restriction says that the absolute value of the maximum eigenvalue of the coefficient matrix in the algorithm have to be smaller that

one, for the system to reach a steady state. We found that it was only the Forward Euler method that needed a restriction on the step length, namely $\Delta t < \frac{\Delta x^2}{2}$. This restriction was used as a unit test in the algorithm of Forward Euler.

We also had to find a way to test the two other methods. We showed analytically that there was no restrictions on the step lengths for these, and this result came from the fact that the coefficient matrix of their algorithm do not have any eigenvalues with absolute value bigger that or equal to one. To double check that this is the case we find the eigenvalues of the coefficient matrices numerically, and make sure that the absolute value of all the eigenvalues of the matrices are less than one. This unit test was also used on the Forward Euler method, to make sure that the restriction we found on the step length was right.

3.3.2 Verification of results

In 2.1 we found the analytical solution of the diffusion equation that we are looking at in this project. All the numerical methods that we are using in this project are supposed to give us the same answers, namely the answers that the analytically calculated solution gives. If we compare the results from the numerical methods with the results from the analytical solution, this is a verification of that the numerical calculated results are correct. We can also use the results from the analytical solution to find the relative error in the results from the numerical methods.

4 Result and discussion

4.1 Truncation error and stability properties

Numerical method	**Truncation error**	**Stability requirements**
Forward Euler	$\mathcal{O}(\Delta t)$ and $\mathcal{O}(\Delta x^2)$	$\Delta t \leq \frac{1}{2}\Delta x^2$
Backward Euler	$\mathcal{O}(\Delta t)$ and $\mathcal{O}(\Delta x^2)$	Stable for all Δt and Δx.
Crank-Nicolson	$\mathcal{O}(\Delta t^2)$ and $\mathcal{O}(\Delta x^2)$	Stable for all Δt and Δx.

Table 1: This table lists the order of the local truncation error in space and time for three numerical methods. It also lists the requirements on the step lengths in space and time, to ensure that the numerical methods produce stable results.

4.2 Numerical calculated results

In this section we are studying the density of a particular type of transmitters, graphed as a function of position in the synaptic cleft. As anticipated the transmitters will spread

out from the presynaptic cell, which is the start point, against the postsynaptic cell, which is the end point. This is the work of diffusion. At small times, for example at $T = 0.001$, we can see that there is no transmitters in most of the synaptic cleft. As time goes, the system is approaching equilibrium, and we can see that equilibrium is reached at $T = 0.3$. We can see an example of this in Figure 3.

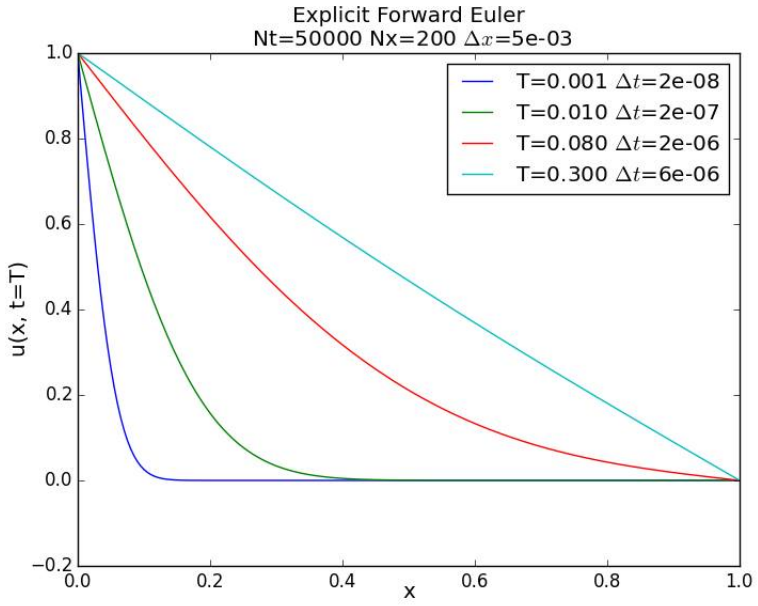
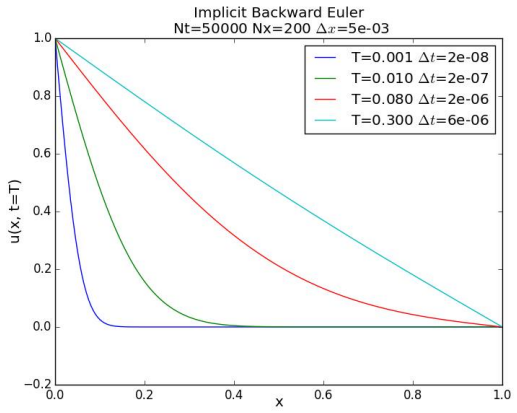


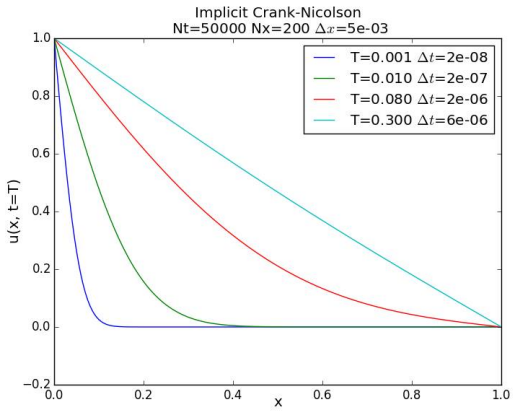
Figure 3: The concentration of neurotransmitters as a function of position in the synapitc cleft for four different points in time. This results is computed by use of the Forward Euler method. Nt is the number of time steps and Nx is the number of step length.

Figure 3, 4a and 4b shows the results from Forward Euler, Backward Euler and Crank-Nicolson. We can see that it is almost impossible to see the difference in the precision of the three methods from these plots. Figure 4c and 4d shows the results from brute force Monte Carlo simulation respectively without and with Gaussian step lengths. Here it is more easy to pick out some differences between the plots. In Figure 4c we can spot some irregularities in the graph for $T = 0.001$, these are not present in Figure 4d. A final remark is that the Monte Carlo simulation with constant step length produces results that in first sight looks like the results from Crank-Nicolson and Forward and Backward Euler. This is in contrast to the results made by the Monte Carlo simulation with step lengths drawn from the Gaussian distribution. In Figure 4d we can see that all the graphs have correct behavior in most parts, but that the density values seems to be a little too small all over. We also see that they have an odd behavior in the start of the x-interval, and in Figure 5a we can see this area zoomed in. The data are scaled so that the value in the initial position is equal to 1. In Figure 5b we can see the points that makes up the graph at $T = 0.3$. It is obvious that the three first points shows some

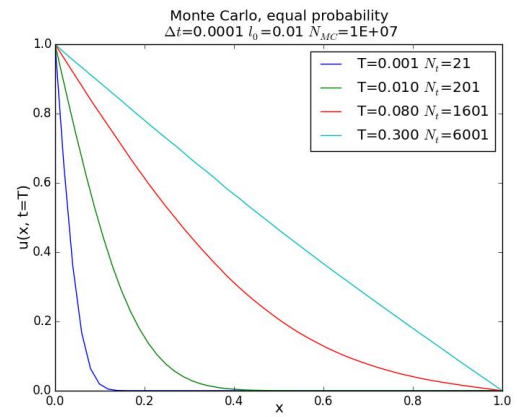
instability, and that the following points follows the expected behavior. From this we can conclude that the density in the initial position is uncorrect, which leads to that the scaling done on all the data is incorrect.



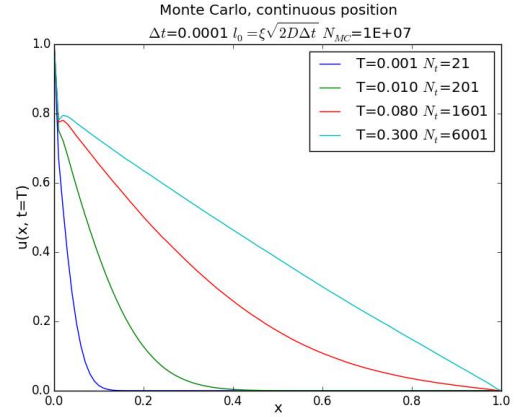
(a) Calculated with Backward Euler.



(b) Calculated with Crank Nicolson.

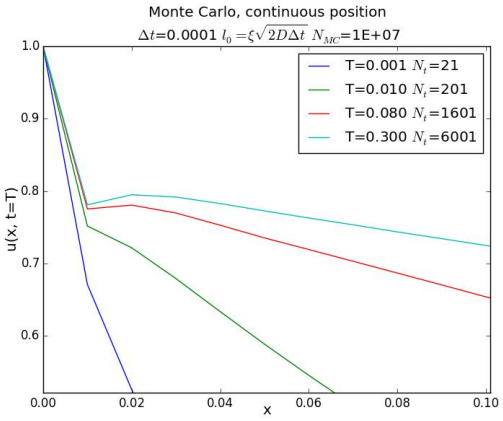


(c) Calculated with brute force Monte Carlo simulation with constant discretized step length.

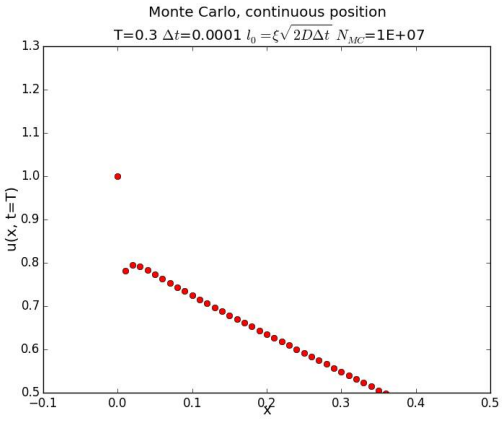


(d) Calculated with brute force Monte Carlo with continuous step length drawn from the Gaussian distribution.

Figure 4: The concentration of neurotransmitters as a function of position in the synapitc cleft for four different points in time. These results are computed by use of the Backward Euler method, Crank Nicolson scheme, brute force Monte Carlo and Monte Carlo with importance sampling. The Gaussian distribution is used in the importance sampling, as the neurotransmitters step length. N_t is the number of time steps, N_x is the number of step lengths, N_{MC} is the number of Monte Carlo cycles and l_0 is the constant step length of the neurotransmitters.



(a) All graphs.



(b) Points from graph $T = 0.3$ (equilibrium).

Figure 5: Figure 4d zoomed in at top left corner.

4.3 Error in numerical calculated results

4.3.1 Relative and absolute global error.

We know from section 2.3 that the spatial local error in the three methods Forward Euler, Backward Euler and Crank Nicolson, are of magnitude $\mathcal{O}(\Delta x^2)$. Although the magnitude of the local error is the same in the three methods, the global error can accumulate to be quite different. We also know that the order of the local error in time is smaller for Crank Nicolson, something that will make the global error less for that method.

In Figure 6 we can see the relative error for the three methods. As we would expect, the relative error is linear. This is because we add the same amount of local error in every time step of the algorithm. We can see that the graphs nearly stops to grow when x is approaching 1.0. This is a consequence of that we have boundary conditions in the initial and final position. This fact is easier to realize if we take a look at the absolute error, which is the absolute value of the difference between the numerically and analytically calculated data. The absolute error for the three methods is graphed for two different times each in Figure 7a. We see that it is approaching zero in the initial and final position, just as expected, because we have fixed the known values of the boundaries in these positions. This again leads to a maximum relative error in the midpoint position, because the calculated density u_j^i in every position x_j at time t_i depends on the positions x_{j-1}, x_j and x_{j+1} at time t_{j-1}. Therefore the behavior of the absolute error is as expected, and we can assume that this also is the reason for that the global relative error stops to grow as it approaches the final position.

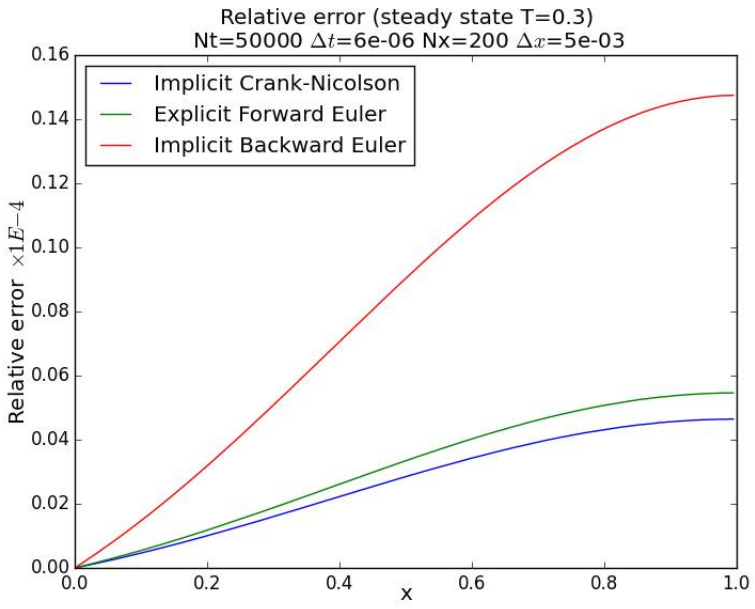
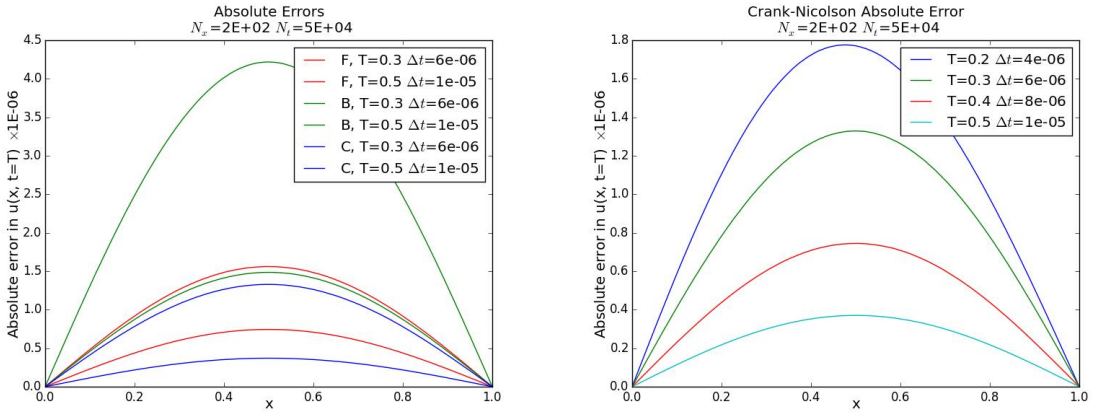


Figure 6: The global relative error of the numerically calculated data, graphed as a function of position in the synaptic cleft. The numerical data are computed by use of Forward Euler method, Backward Euler method and the Crank Nicolson scheme.



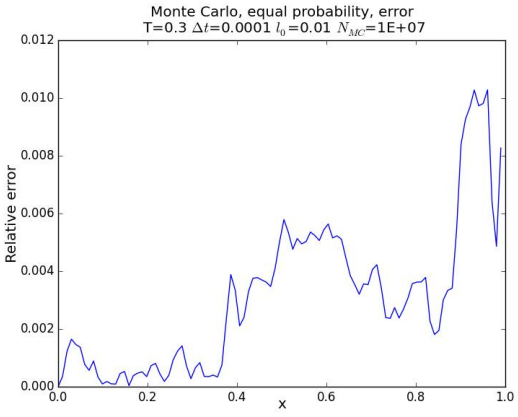
(a) Absolute error of the numerical data computed with Forward Euler (F), Backward Euler (B) and Crank-Nicolson (C), for two difference times each.

(b) Absolute error of the numerical data computed with Crank-Nicolson for four different times.

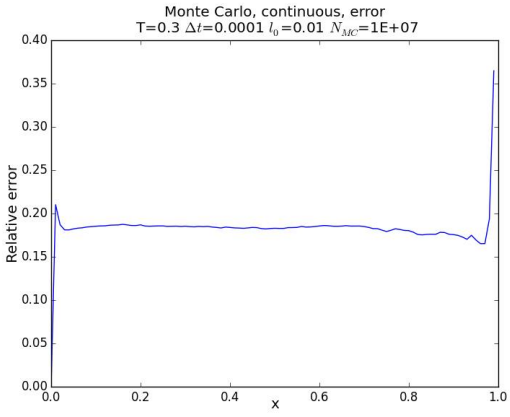
Figure 7: Absolute error of the numerically calculated data, graphed as a function of position in the synaptic cleft.

Because the absolute error in this case reflects the behavior of the accumulated data best, we wanted to analyze it further. As mentioned, in Figure 7a we can see the absolute error of the three methods for two different times each. From this we can see how the absolute error changes with time for the three methods, and we can easily see which method is preferable. Backward Euler has a huge absolute error in one of its graphs compared to the two other methods. We can also see that the absolute error of Forward Euler starts out higher than that of Crank-Nicolson, and that it keeps that way for the time that gives the best results. Then we see that the Crank-Nicolson scheme is giving the best accuracy of the three methods, as expected. In Figure 7b the absolute error of the numerical data computed with Crank-Nicolson is graphed at four different times, so that we can see how it is evolving in time. We can see that the absolute error is getting smaller as the time gets higher. From the results in 4.2 we can see that it looks like the system has reached equilibrium at $T = 0.3$, but we can see from Figure 7b that the error gets smaller and smaller as the time goes, so that the system gets even closer to equilibrium. This is not a fact that we can take for granted, because the the number of steps in time is constant for the four different time interval, which means that the step lengths of the interval with final value $T = 0.5$ is bigger than the step lengths of the other time intervals. Even so, from Figure 7b we can conclude that a time interval with a big final value T_h will give a better precision than a time interval with a lower final value T_l.

In Figure 8a and 8b we can see the relative error of the data accumulated from Monte Carlo simulation with constant step length and with continuous step length drawn from the Gaussian distribution respectively. After what we discussed in 4.2 it is no surprise to see that the relative error in Figure 8b is higher than the relative error in Figure 8a. The interesting thing here is to compare the relative error in Figure 8a with the relative errors in Figure 6, since the resulting plots of these methods in 4.2 seems to be quite satisfactory. Still, we can see that the relative error in the Monte Carlo method is around 10^3 times bigger than the relative error in the three PDE-solvers. From this we can conclude that the Monte Carlo simulation in this case is wasted, when we can use the three other methods. The Monte Carlo simulation is though a more intuitive method in this case, because random walks and diffusion is closely linked.



(a) Monte Carlo with constant step length.



(b) MonteCarlo with continuous step length drawn from the Gaussian distribution.

Figure 8: Relative error of the numerically calculated data, graphed as a function of position in the synaptic cleft. The data are computed by use of brute force Monte Carlo simulation and Monte Carlo with importance sampling. The relative error is computed at time $T = 0.3$.

5 Conclusion

During this project we have analyzed the results generated from five different methods of solving the diffusion equation. We have learned that the Crank-Nicolson method theoretically should give results with better accuracy than Forward and Backward Euler, and we have showed that this also is the fact when solving the diffusion equation. We saw that both Forward and Backward Euler, and then of course Crank-Nicolson, gives better results than the Monte Carlo simulations with discrete and continuous step lengths. The Crank-Nicolson scheme has a accuracy that is $\sim 10^3$ times better than that of the most accurate Monte Carlo simulation. From this we conclude that there is no need for Monte Carlo simulation when solving the diffusion equation in one dimension, because the Crank-Nicolson scheme gives much better results.

6 Comments

The code used to generate the results in this project can be retrieved on web page `<https://github.com/vildemf/compphys/tree/master/project5>`.

References

[1] Morten Hjorth-Jensen, Computational Physics, Fall 2015, Department of Physics, University of Oslo

[2] Knut Mørken, Numerical Algorithms and Digital Representation, August 2013, Department of Mathematics, Centre of Mathematics for Applications, University of Oslo

[3] <https://en.wikipedia.org/wiki/Taylor_series>

[4] <https://en.wikipedia.org/wiki/Fourier_series>

[5] <https://en.wikipedia.org/wiki/Tridiagonal_matrix>

[6] <https://en.wikipedia.org/wiki/Toeplitz_matrix>